NTNU
Norwegian University of
Science and Technology

Hybrid Modelling for Dynamic Reliability Assessments in Subsea Oil and Gas Processing Systems

Renny Jose Arismendi Torres

Reliability, Availability, Maintainability and Safety (RAMS)

Submission date:	June 2017
Supervisor:	Mary Ann Lundteigen, MTP
Co-supervisor:	Anne Barros, MTP
	HyungJu Kim, MTP
	Andreas Hafver, DNV-GL

Norwegian University of Science and Technology
Department of Mechanical and Industrial Engineering

Preface

This thesis is part of the requirements of the international master program in RAMS (Reliability, Availability, Maintainability and Safety) within the Department of Mechanical and Industrial Engineering (MTP) at the Norwegian University of Science and Technology (NTNU). It has been carried out during the spring semester of 2017.

The project proposal came from the center for innovation-driven research on subsea production and processing, SFI SUBPRO, suggested by one of the industry partners of the center: DNV GL. This project has been carried out in collaboration with DNV GL.

The report has been written for readers with some basic background knowledge in the field of safety and reliability analysis.

Renny Arismendi

Renny Arismendi Torres

Trondheim, 11 Jun. 2017

Acknowledgment

I would like to express my gratitude and appreciation to my supervisors, Professor Mary Ann Lundteigen, Professor Anne Barros, Postdoc HyungJu Kim and DNV-GL Andreas Hafver for all the patient guidance, feedback, active encouragement and motivation throughout the project. I feel that together, this strong team kept me going in the right direction.

I would also like to thank my classmates of the master program, with whom I had many talks and discussions about each other's projects, helping to have a better understanding of the topic, and making the learning process a more enjoyable experience.

Summary and conclusions

Traditional reliability methods such as fault tree analysis and reliability block diagrams were not designed to account for the complex dynamic behavior of most industrial systems. These traditional reliability methods make simplifications in the modelling of many engineering systems, by assuming static operational and environmental conditions. Therefore, they are not able to account for the influence of the dynamics of the operations, environment or interactions between components of the system over the failure rates or degradation process of these, neither for the reciprocal influence of the failures and degradation processes over the process and system performance. For this purpose, dynamic reliability has been conceived as an extension of traditional reliability methods, by combining these with process modelling, automation and the theory of dynamic systems.

This thesis focuses in hybrid modelling, as a mean to approach dynamic reliability problems. In hybrid modelling, the problem is described as dependent processes that are modelled separately and then coupled by means of shared variables, describing the dynamic behavior of the system in terms of deterministic physical laws and stochastic models describing failures and/or degradation processes.

Driven by the goal in the oil and gas industry of finding new solutions for moving entire topside processing facilities to the seabed, a separator case study is presented, exploring the potential of hybrid models and demonstrating the modelling and simulations procedure and characteristics. The dynamic hybrid model of the case study is constructed using Simulink® as a tool for conducting simulations. Discussions about the modelling procedure and the hybrid model capabilities are included, pointing at the potential and challenges or drawbacks.

The results show that hybrid models allow to integrate systems design with reliability analysis, from early stages of new technology development, giving a mean to find optimal solutions regarding safety, cost and performance, while providing confidence in the qualification of such technology. Dynamic reliability assessments are meant to fill the areas in which the traditional methods fall short, complementing these rather than replacing them.

Contents

Chapter 1

Introduction

1.1 Background

During the past decades, offshore developments have been moving into deeper waters and areas more difficult to access, assisted by the installation of subsea production and processing systems. The vision for the future of the oil and gas industry is to gradually move entire topside processing facilities to the seabed, increasing the recovery and aiming for a Subsea Factory™, a concept developed by Statoil (2014). Projects that previously were considered non-commercial could become economically viable and existing fields can increase their recovery rates.

The installation of processing facilities on the seabed also comes with associated hazards and risks that are in principle different from those on topside, due to different environment and operating conditions, and hence, this new technology introduces uncertainties entailing risks to its developers, manufacturers, vendors, operators and end-users. The development of new technology shall go through a process of qualification, providing evidence that it will function within specified limits of operation with a confidence level that can be considered acceptable (DNV-RP-A203 (2011)).

Processing systems have a complex dynamic behavior, in which the components of the system interact with each other in dynamic environmental and operational conditions. As pointed by Babykina et al. (2016), traditional reliability assessment approaches such as fault tree analysis and reliability block diagrams were not designed to account for the dynamic structure function of systems of this type. These traditional approaches are usually applied supported by the practice of thinking of the worst scenario, as pointed by Chiacchio et al. (2016b).

To account for the complexity of processing systems, more elaborated methods, capable of capturing the dynamic behavior of these systems in an accurate way, are required to evaluate their performance and asses their reliability. Dynamic reliability is an extension of the traditional reliability and dependability theories, combining them with process modelling, automation and theory of dynamic systems (Manno et al. (2015)). One way to approach dynamic reliability problems is through hybrid modelling; a formalism that has been proposed in previous works by Manno et al. (2015), and has shown the potential to approach this type of problems in an efficient and systematic manner.

Hybrid modelling allows to couple process deterministic models with reliability stochastic models, in a way that stochastic transitions between various states of operation affect the solution of the deterministic model and vice versa. Hybrid models may allow to identify how different sequences of component failures can lead to failure modes of the overall system or to critical events, and how degradation of components may affect the performance of the overall system in a quantitative way. Hybrid modelling may provide a mean to qualify profitable new technology solutions, relevant for Subsea Factories™, as well as for integrating reliability analysts with systems designers at early stages of the technology development, extending the RAM techniques to account for the impact of the dynamic behavior of the systems in production constraints other than reliability or availability issues.

This project focuses on subsea separation, a key research area for new subsea technology developments, to build a case study, investigating the potential of the application of hybrid models in new technology development and qualification.

1.2 Objectives

The main objective of this master thesis is to investigate the procedure and potential of the application of hybrid modelling in the context of subsea oil and gas processing, for new technology development and qualification. For this purpose, the following sub-objectives are defined:

1. Study and present the concepts of dynamic reliability and hybrid modelling.
2. Identify, select and describe a relevant subsea separation case study system, for the application of hybrid modelling for dynamic reliability assessments.
3. Study and present the subsea separation process and dynamic model.
4. Study failure mechanisms and modes of the system, such as degradation of components and present stochastic models of these.
5. Construct a hybrid model of the system, coupling the deterministic and stochastic processes models using an appropriate tool for conducting simulations.

6. Explore how the hybrid model may be used in reliability and performance assessments for the development and qualification of new technology.

1.3 Limitations

Dynamic reliability problems and hybrid models may require advanced computer science skills for the ability to model it accurately and run simulations efficiently. As pointed by Chiacchio et al. (2016a), at the state of the art, no tools or computer-aided system have been coded to solve this type of problems. This fact, combined with lack of advanced programming skills may limit the complexity of the dynamic process that can be modelled and simulated.

During this project, the only tool explored for simulations of the dynamic hybrid model is Simulink® and while it is a suitable tool for this type of problems, no other tools have been explored. Hence, the discussions around the modelling and simulations and the corresponding challenges encountered are limited to this tool, and no comments can be made about other available tools.

In this work, hybrid modelling is introduced and formulated in a general manner, as a mean to include different modelling formalisms available in the literature under the same framework. Formalisms such as piecewise deterministic Markov process or switching diffusion process are part of the general hybrid modelling as proposed in this work, but are not studied in detail. Therefore, no comments or discussions on specific formalisms are included in this work, but discussions are made from a rather high level viewpoint over hybrid models as general.

The separator case study presented in this work is not based on an existing system, and the parameters used in the models for process, control and degradation, are not supported by experimental data. The case study does not aim to solve a specific problem or to draw particular conclusions for the separation process or control model, but to demonstrate the approach, potential and challenges of hybrid modelling for dynamic reliability and performance assessments.

1.4 Research approach

1.4.1 Literature study

Literature review serves the basis for the project. The main research topics for literature review are hybrid systems, dynamic reliability, subsea separation process and control, and technology qualification.

Literature research is done through available common scientific databases as ScienceDirect, SCOPUS, Springer link and others for scientific papers, OnePetro is used for technical literature for the oil and gas industry, Standard.no and IHS Standard expert are used for consulting relevant standards, in addition to the university's libraries and consultancy to experts in the relevant topics to be studied.

1.4.2 Tools and methods

Because of the technical complexity of the topic under study, and the fact that many publications in the topic are specific to an application, method and/or tool, the learning process from a literature review approach can be a challenging and tedious task. For this reason, it was decided that the most suitable approach to gain knowledge and a good understanding of the topic under study is through experiential learning. Experiential learning is the process of acquiring knowledge through experience, or learning through "reflection on doing".

Under this reasoning, it was decided to place the focus in a few relevant papers on the topic and case studies, as a starting point. Case studies from previous works by Manno et al. (2013), Manno et al. (2015), Nguyen et al. (2015), Langeron et al. (2012), Langeron et al. (2015) are chosen to be reproduced and modelled with the chosen tool. This allows to gain knowledge and skills with the modelling procedure, the tool, and a deeper understanding of dynamic reliability problems, that is later complemented and supported with more literature review, as new questions arise or for clarification of some points. It also allows to identify the relevant factors or elements to look for when setting up the subsea separator case study and to frame the literature research for this purpose.

The tool selected for modelling and simulations is an integration of Simulink® and MATLAB®. These tools have been used for conducting dynamic reliability analysis in previous works and are available to all students at NTNU, making it a convenient choice. Auto didacticism is used for the modelling tools Simulink® and MATLAB®. For this, Google is the most powerful and friendly search engine, giving access to many forums and discussions, besides the official documentation and help page of the developer.

The separator case study presented, as well as case studies and literature from previous works, are the base for reflections and discussions over the modelling approach, potentials and challenges. Based on these discussions, suggestions are made to hybrid modelling and simulations, attempting to keep the approach in a general formulation, capable to include different kind of problems of interest for the RAMS field.

Regular meetings are carried out with supervisors of the project for discussions around the findings, model progress, future works, etc. Group meeting as well as individual meeting are conducted according to the specific purpose of the meeting and fields of contribution of the supervisors involved.

1.5 Structure of the report

The report of the master thesis is structured in chapters. Chapter 2 introduces the concepts of dynamic reliability and hybrid models, elaborating on the modelling formalism and simulation algorithms from a general point of view. Chapter 3 introduces the concept of subsea separation process and models, framing the case study. Chapter 4 describes failure mechanisms identified of the separation system, and models to describe them as stochastic processes. Chapter 5 details the construction of the hybrid model of the separator case study, detailing how the coupling between deterministic and stochastic models is achieved, and an overview of how the model is implemented in Simulink® for simulations. Chapter 6 elaborates on the potential of application of hybrid models for reliability and performance assessments in new technology development and qualifications processes, and discusses on the challenges or drawbacks. Chapter 7 presents a guideline to hybrid modelling for dynamic reliability applications. Chapter 8 presents the summary and conclusions of this thesis and proposes recommendations for further work.

Chapter 2

Dynamic reliability

The theory of hybrid and stochastic hybrid systems is useful for introducing concepts of dynamic reliability. This chapter elaborates on the concept, the modelling and methods found in the literature for dynamic reliability cases and attempts to create a general knowledge involving those cases under a common framework, taking bases on the developments in hybrid systems theory. Hybrid systems, as pointed out by Lunze (2002), have been a major research topic of the past decades, attracting the interest of mathematicians, control engineers and computer scientists, and the methods applied and results obtained are as diverse as the background of the researchers. Findings in this topic are of interest for expanding the models and methods of traditional reliability, in order to account for the dynamic conditions in which engineering systems evolve in safety and reliability analysis. The concept, models and methods of dynamic reliability are presented from a general point of view.

2.1 Hybrid systems

In general, a hybrid system is understood as a dynamic system with both continuous and discrete dynamic behavior. Lunze (2002) defines a hybrid system as “a dynamical system that cannot be represented and analyzed with sufficient precision either by the methods of continuous systems theory or by the methods of the discrete systems theory”.

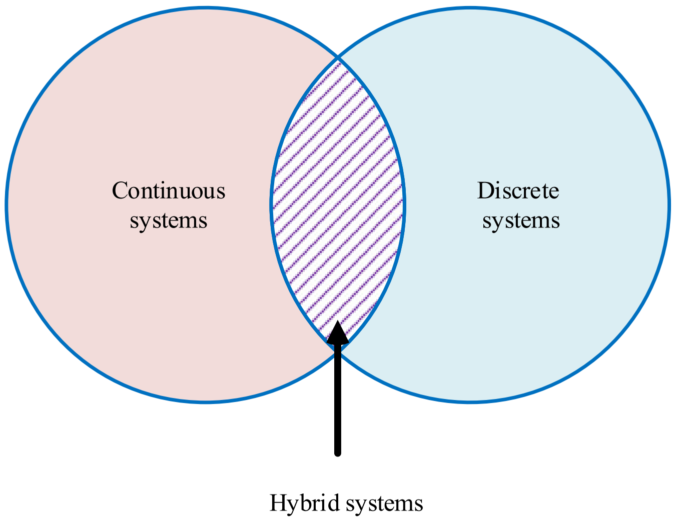


Figure 2-1. Hybrid systems.

Hybrid systems have existed for long but as pointed by Lunze (2002), previous to the appearance of the currently developing theory of hybrid systems, these systems have been considered as purely continuous or purely discrete. Because of the fact that many modern technological, industrial or engineering processes cannot be clearly analyzed and controlled by investigating only the continuous or only the discrete behavior, hybrid system have become a hot topic in research.

The theory of hybrid systems relies on combinations of continuous and discrete systems theory in order to describe the dynamics of a system.

Continuous systems theory

Under continuous systems theory, the system under consideration can be described by a differential equation of the type:

$$\frac{\partial x}{\partial t} = f[x(t), u(t), t], \qquad x(0) = x_0 \tag{2-1}$$

$$y(t) = g[x(t), u(t), t] \tag{2-2}$$

Where:

- t denotes time,
- $x(t)$ is the state vector and x_0 is the vector of initial state,
- $u(t)$ is the input vector,

- $y(t)$ is the output vector,
- f and g are the vector functions that describe the system behavior.

In a more general form, equation (2-1) can be replaced by a set of differential and algebraic equations known as a differential-algebraic system.

Discrete systems theory

Discrete systems theory considers systems with countable number of states. The signals of these systems are assumed to have values from a finite or infinite discrete value set. Under this assumption, the system can jump from a discrete state to another but the continuous trajectory of the system cannot be described. The sudden changes of states are called events.

Hybrid system

A hybrid system is then a dynamical system that when subject to a continuous input $u(t)$, it follows a behavior that results from a combination of continuous state changes and sudden state jumps (events). The continuous and discrete dynamics do not only coexist in a hybrid system, but they can interact in a way that the evolution of continuous variables influence the discrete variable dynamics and vice-versa.

Modelling of modern industrial and engineering complex systems require uncertainties to be taken into account for proper assessments. Uncertainties in the choice of continuous evolution, uncertainties in the choice of the state of destination or timing of a discrete jump need be quantified probabilistically for the development and operation of safe, reliable and maintainable modern systems.

Incorporating the theories of probabilistic and stochastic concepts into hybrid systems, introduces another class of systems, known as stochastic hybrid systems.

2.2 Stochastic hybrid systems

Stochastic hybrid systems combine hybrid systems with the theory of probabilities to deal with randomness. A stochastic hybrid system is a dynamical system with both continuous and discrete behavior in which some variables cannot be described in a deterministic way, but they may evolve in different ways, i.e. a unique input does not lead to a unique output.

As mentioned by Bujorianu (2012), the existing techniques, methodologies and tools for design, control, analysis, simulation and verification for hybrid models have to be redesigned from scratch when dealing with randomness, in order to consider probabilities. This fact makes it necessary to distinguish between hybrid systems and stochastic hybrid systems.

Figure 2-2 shows the relation between continuous, discrete, hybrid and stochastic hybrid systems.

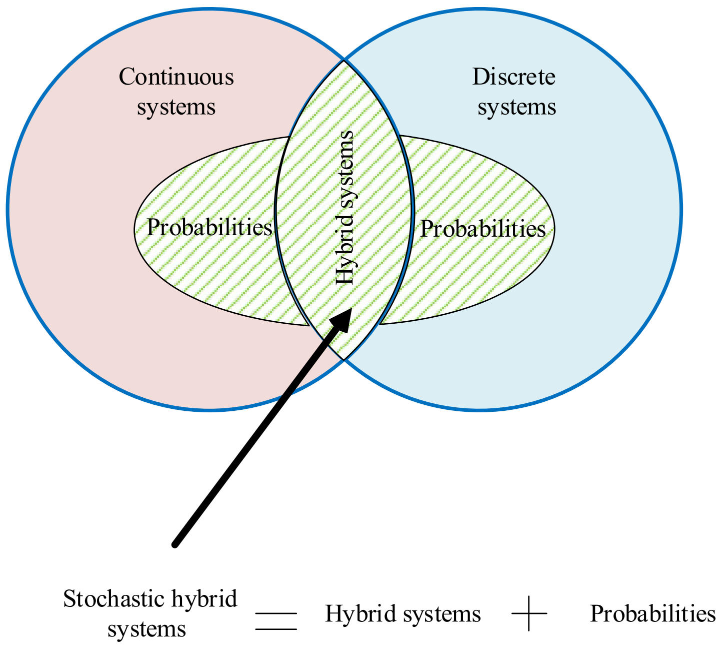


Figure 2-2. Stochastic hybrid systems.

Randomness and hence stochastic hybrid systems are of particular interest in the RAMS field, where system safety and reliability analysis rely on probability distribution functions of a time to failure, to account for uncertainties of influencing factors such as operating conditions, loads and strength of components.

Rausand et al. (2004) present a general definition from the standard ISO 8402, which states that reliability is "the ability of an item to perform a required function, under given environmental and operational conditions and for a stated period of time". The word "item" refers to any component, subsystem or system that can be considered an entity. This definition is widely accepted and slight adaptions to it may be made for specific types of systems. Clearly, this traditional definition of reliability does not account for the changing environmental and operational conditions in which most of the industrial systems evolve. Traditional systems reliability theory, models, methods and applications, have been conceived over this definition of reliability and as result, were not designed to account for the dynamic evolving conditions.

Based on this and supported by the concept of stochastic hybrid systems, the framework of dynamic reliability can be introduced.

2.3 Dynamic reliability concept

Although different methodologies have been investigated and developed over recent years to incorporate process dynamics and stochastic transitions, a clear framework placing all these methodologies in a common group is not clearly found in the literature. As pointed by Labeau et al. (2000), despite their potential, dynamic reliability methods have not penetrated the field of industrial applications and no defined effort has been taken to create a generic platform for performing dynamic reliability analysis; practical applications to industrial systems are limited and the existing ones have been mostly conducted by experts using application-dependent software. In the current literature, there are several contributions from systems experts devoted to distinct research aspects, but at the state of the art, there are no integrated platforms that are able to combine all these aspects, as indicated by Chiacchio et al. (2016a). This makes it difficult to find a standardized or universally accepted definition of dynamic reliability.

Based on the traditional definition of reliability presented in section 2.2, it is reasonable to think of dynamic reliability as the ability of an item to perform its required function for a stated period of time, considering the interactions with the dynamic environmental and operational conditions. Liu et al. (2015), define dynamic reliability as "the reliability function or model of a specific system that can be updated or modified by collecting additional useful knowledge or information related to the stochastic (deteriorating) behaviors of the system after the system is put into use".

As pointed by Manno et al. (2015), dynamic reliability is an extension of classical dependability theory that was conceived because of the need to achieve more realistic reliability assessments in the field of nuclear engineering and in general, in fields that are characterized by major hazards. It is a combination of classical reliability theories with process modelling, automation and theory of dynamic systems.

Combining classical reliability theories with process modelling, automation and/or dynamic systems, requires models in which some variables evolve in continuous form, others may exhibit discrete behavior, and probability theories are included to deal with uncertainties or randomness. It is reasonable to think of dynamic reliability as reliability of stochastic hybrid systems. It is worth pointing out that although the literature refers to stochastic hybrid systems, the term "systems" refers to models and not to engineering or industrial systems as such. In this sense, exploring the potential of available stochastic hybrid systems formalisms for safety and reliability analysis of engineering systems can create the framework of dynamic reliability.

Dynamic reliability aims to extend traditional reliability models and methods, with ones that are capable of capturing the dynamics of the operational and environmental conditions in which systems evolve, as well as the dynamics of interactions between components of the system by combining random discrete events such as failures with continuous and deterministic processes.

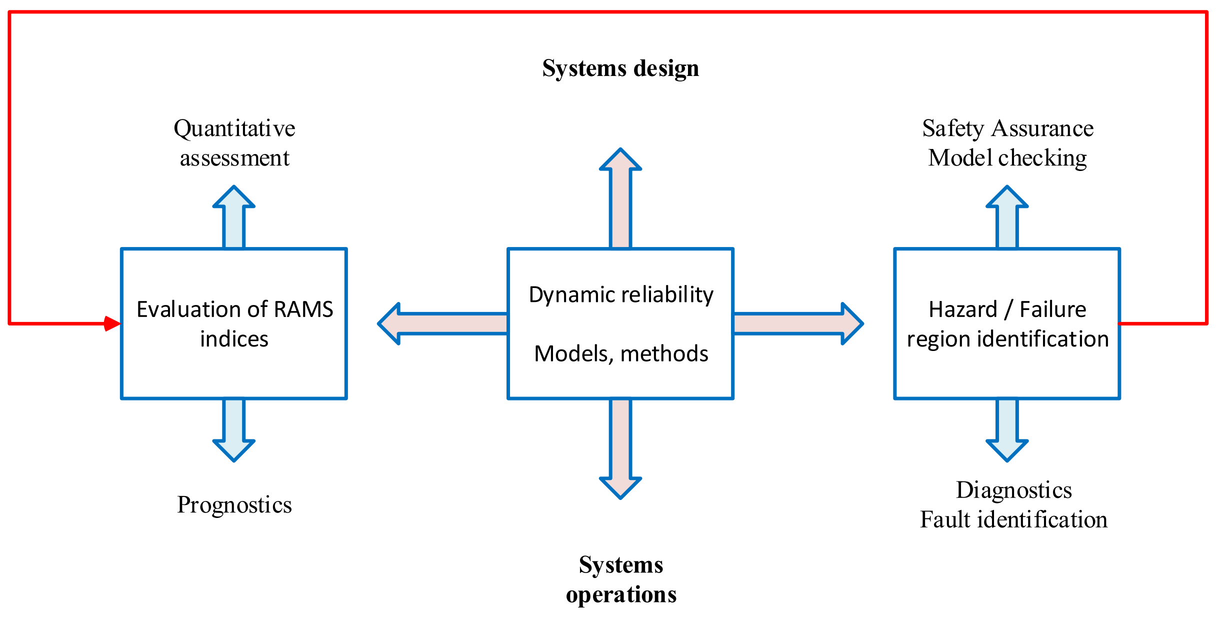

Systems design
Quantitative assessment
Safety Assurance Model checking
Evaluation of RAMS indices
Dynamic reliability
Models, methods
Hazard / Failure region identification
Prognostics
Diagnostics Fault identification
Systems operations

Figure 2-3. Dynamic reliability applications. (Adopted from Manno et al. (2015))

The models and methods of dynamic reliability find applications in systems design and operations as suggested by Manno et al. (2015) and shown in Figure 2-3, similar to the traditional reliability approaches.

Dynamic reliability problems are not an easy task to deal with. Challenges arise in the modelling of a dynamic system, in a way that the model is able to capture the evolving nature of such system and describe it by means of continuous, discrete and random variables.

2.4 Hybrid modelling

2.4.1 Hybrid systems modelling

Kowalewski (2002) indicates that there are in principle three different ways to create a model for hybrid systems:

1. Take existing discrete formalisms and extend them by incorporating continuous variables that evolve according to differential equations associated with discrete states. The discrete transitions can switch between continuous modes and the continuous variables can be reset when a transition takes place. Examples of this approach of hybrid modelling consist of extending formalisms such as finite automata or Petri nets to hybrid automata or hybrid Petri Nets.

2. The second approach is the opposite direction of the first approach, i.e. extend models for continuous systems with discrete mechanisms such as switching or resetting time dependent continuous variables. The resulting framework consists of differential, algebraic equations and inequalities with both continuous and binary variables. The binary variables can be used to activate or deactivate terms of the equations. This class of systems is referred to as switched continuous systems.
3. The third approach consist of not extending a formalism but to use an existing continuous model and an existing discrete model as they are, and couple them by means of appropriate interfaces. This approach allows usage of commercial simulation tools like Simulink®.

Although Kowalewski (2002) refers to hybrid systems and not to stochastic hybrid systems, the same approaches can be applied for modelling stochastic hybrid systems by incorporating random variables.

2.4.2 Stochastic hybrid systems modelling

Hu et al. (2000) mentioned different ways of introducing randomness into hybrid systems that one can relate to the three approaches indicated by Kowalewski (2002):

1. For the first approach, which builds over existing discrete formalisms, an obvious choice is to replace the deterministic jumps between the discrete states by random jumps governed by a probabilistic law. This leads to formalisms such as stochastic hybrid automata and Piecewise-deterministic Markov processes (PDMP), an extension of a Markov process with continuous states incorporated with evolutions that follow state-dependent deterministic differential equations.
2. In the second approach, which build over existing continuous systems, the choice is to replace the deterministic dynamics inside the invariant set of each discrete state by stochastic differential equations. Even while keeping the deterministic discrete transitions, different guards can be activated depending on the realization of the stochastic process, causing discrete transitions that occur randomly. A framework of this type is switching diffusion processes (SDP).
3. A third choice allows to model continuous and discrete systems separately using existing formalisms, incorporate randomness in either or both of them and couple them by means of the appropriate interfaces. It can be understood as a blending of both the first and second approach that can lead to a more general definition of stochastic hybrid systems.

Table 2-1 shows characteristics of the stochastic hybrid systems modelling frameworks, mentioned in the first two approaches.

Table 2-1. Classifications of stochastic hybrid systems. (Adopted from Lygeros (2007)).

	Piecewise deterministic process (PDP)	Switching diffusion process (SDP)
Spontaneous transitions	Covered	Not covered
Stochastic continuous evolution	Not covered	Covered

The intention of the proposed third choice is to combine modelling frameworks from these two approaches, in order to define stochastic hybrid systems in a more general way, i.e. allowing to incorporate randomness in the continuous system, in the discrete system or in both.

In previous work, Manno et al. (2015) proposed a breakdown of a dynamic reliability problem, following a separation of concern approach, into two dependent process, a stochastic and a deterministic one, and referred to it as hybrid-pair modelling.

2.4.3 Hybrid-pair modelling in dynamic reliability

Hybrid-pair modelling is an approach to solve a dynamic reliability problem based on a "separation of concern" approach presented by Manno et al. (2015). The idea behind this modelling approach is to model two mutually dependent processes: a deterministic and a stochastic, separately, and then to couple both models using the shared variables. In this sense, the point of view of continuous vs discrete systems presented in 2.4.2 is shifted to a deterministic vs stochastic perspective for dynamic reliability purposes, but the underlying formulations are the same.

Under this formulation, the deterministic process describes the dynamic system in terms of thermofluid, chemical, rotational and/or other laws that determine its physical behavior, by a set of partial differential equations as (2-1) and (2-2). The stochastic process accounts for events as components failures that can modify the physical behavior of the system described by the set of differential equations, and in vice versa, the rates of transition between stochastic states, can depend on the evolution of the continuous variables described in the deterministic model.

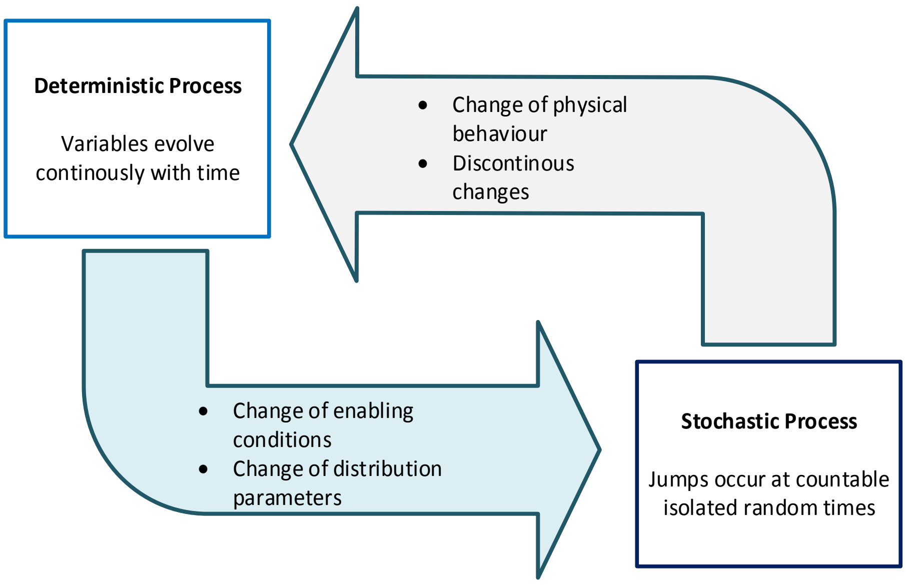

Deterministic Process
Variables evolve continously with time
Change of physical behaviour
Discontinous changes
Change of enabling conditions
Change of distribution parameters
Stochastic Process
Jumps occur at countable isolated random times

Figure 2-4. Coupling of deterministic and stochastic processes. (Adopted from Manno et al. (2015))

Figure 2-4 shows the communication between the two processes as proposed by Manno et al. (2015):

- A change of state in the stochastic process can cause changes in the initial conditions and/or parameters of the deterministic process.
- A change of the state in the deterministic process can change the rate of transition for the next stochastic event.

Generalization and classification

With the purpose of generalizing the hybrid-pair modelling approach presented by Manno et al. (2015) and based on studied concepts of stochastic hybrid systems, a few proposals are made:

1. The modelling approach should allow more than two processes to be modelled and coupled, therefore, the word "pair" and the limitation to one deterministic and one stochastic process is not needed. For example, modelling a dynamic system can include various independent stochastic processes regarding different component failures to be coupled with the deterministic physical behavior of the system.
2. Allow for the inclusion of stochastic processes that evolve as continuous instead of only stochastic processes consisting of discrete jumps. This gives the ability to incorporate continuous degradation processes into the hybrid model. This is very relevant for safety and reliability analysis, for cases of failures that can be predicted by condition monitoring indicators, opening possibilities to assess the performance of a system under the presence of continuous

degrading components. Note that models composed of a continuous deterministic process and continuous stochastic process are not considered a hybrid system from the continuous/discrete perspective, but they represent a type of problem of interest for dynamic reliability for the reasons mentioned and can be considered hybrid from the deterministic/stochastic point of view.

3. Clarify that the deterministic and stochastic processes do not necessarily need to be "mutually dependent", allowing for one-direction dependency, i.e. a deterministic process may be influenced by the evolution of a stochastic process or a stochastic process can be influenced by the evolution of deterministic variables, but the closed loop of dependency is not a requirement of the hybrid model. In this sense, systems may be classified according to its coupling characteristic into two subgroups: tight-coupling (mutually dependent processes) and soft-coupling (one-direction dependency). The soft-coupling subgroup can also be divided according to the direction of the influence or dependency: stochastic to deterministic and deterministic to stochastic.

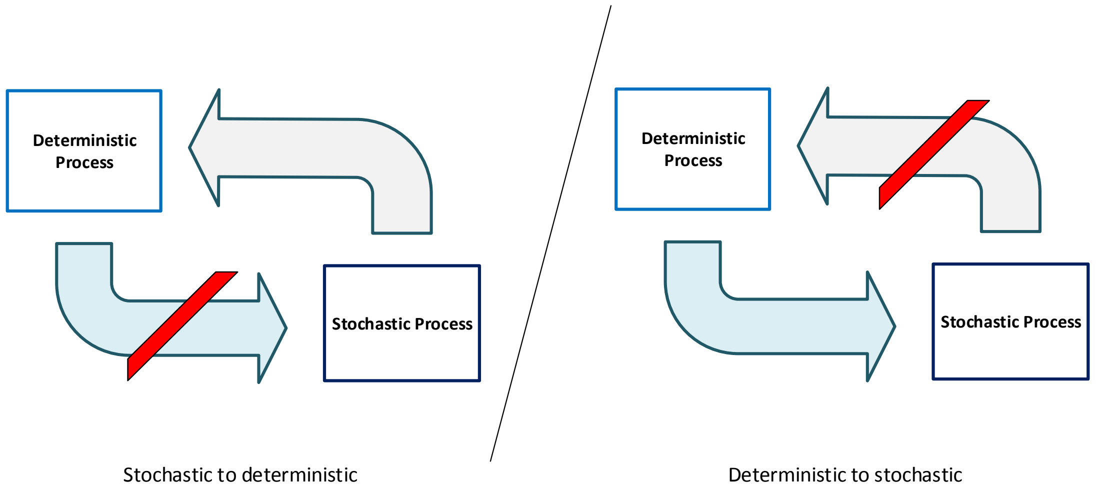


Figure 2-5. Soft-coupling directions in hybrid modelling.

Figure 2-5 illustrates the suggested term of soft-coupling, in which one process influences the evolution of another process, but the mutually dependency is not present. Note that when more than two processes are present in the model as mentioned in suggestion 1, different types of couplings can be present in the model.

It is not always simple to model a stochastic process as dependent on the evolution of continuous deterministic variables. In traditional safety and reliability analysis, failures and degradation processes are modelled as random because the uncertainty around them makes it a very difficult task to link them to a deterministic process. It can still be of interest in dynamic safety and reliability analysis to investigate how failures of components change the dynamics and performance of the overall system, even if these failures are modelled as a purely stochastic process that is not dependent on the evolution of the deterministic process.

With the three suggestions, the approach presented by Manno et al. (2015) can be expanded, giving the opportunity to include more types of problems of interest in dynamic safety and reliability analysis while keeping the same modelling approach.

A hybrid model, even when composed of only two processes, results in complex analytical formulations. For this reason, as pointed by Manno et al. (2015) a dynamic reliability problem can be solved effectively through simulations, with the advantage of overlooking how mathematically complex the model can be.

2.5 Simulations in dynamic reliability

The aim of dynamic reliability is to expand traditional safety and reliability analysis approaches in order to incorporate the dynamic operational and environmental conditions. Considering that these dynamics are described in a deterministic way, in terms of mechanical dynamics, fluid dynamics, thermodynamics and others, several types of hybrid models can be distinguished based on the type of coupling, its direction and the type of the stochastic process. Table 2-2 shows the classification. This distinction allows to consider different simulation procedures.

Table 2-2. Hybrid models classification for dynamic reliability.

Type of coupling	Type of stochastic process
Tight (both directions)	Continuous
Soft: deterministic to stochastic	Discrete
Soft: stochastic to deterministic	

In their work, Manno et al. (2013) proposed a discrete event simulation algorithm for their formulated type of hybrid model, which is characterized by a tight-coupling and discrete stochastic process. This simulation procedure can be used for hybrid models with a discrete stochastic process, but a simplification of the algorithm can be made for hybrid models with only soft-coupling in stochastic to deterministic direction. For hybrid models with a continuous stochastic process a much simpler continuous simulation approach can be used.

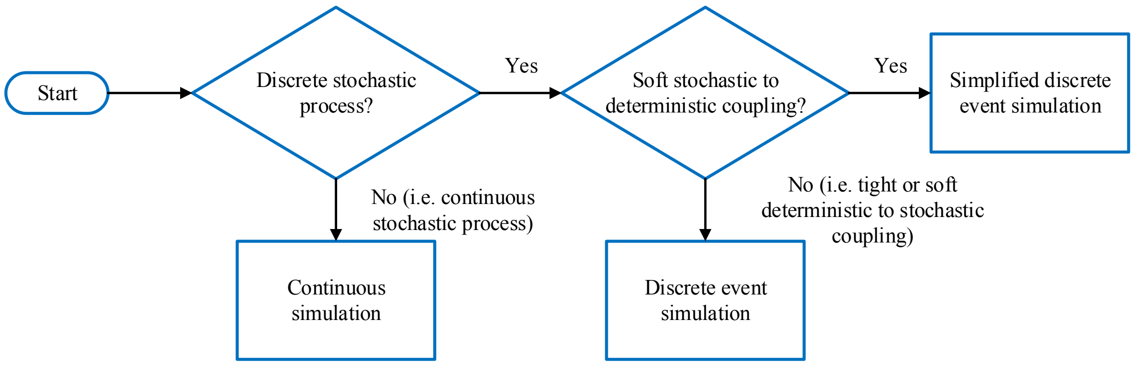

Start
Discrete stochastic process?
Yes
Soft stochastic to deterministic coupling?
Yes
Simplified discrete event simulation
No (i.e. continuous stochastic process)
Continuous simulation
No (i.e. tight or soft deterministic to stochastic coupling)
Discrete event simulation

Figure 2-6. Types of simulation for hybrid models in dynamic reliability

Figure 2-6 shows the choice of simulation type based on the classifications of the model proposed in Table 2-2.

2.5.1 Discrete event simulation

A discrete event simulation algorithm for solving a dynamic reliability problem can be formulated as proposed by Manno et al. (2013), in the following way:

Formulation

Let

- s - state of the stochastic process.
- *d* - state of the deterministic process.

For every transition between different stochastic states, the following are defined:

- *act* – function of activation of the given transition.
- *p* – parameter of the given transition.

Such that, *act* and *p*, may both depend on *s* and *d*. So, *act(s,d)* and *p(s,d),* are the functions which serve as the mechanism of communication from the deterministic to the stochastic process.

The communication from the stochastic to the deterministic process, is achieved by taking *s,* as an additional algebraic variable into the set of differential equations (2-1) and (2-2). It can be used to:

- Change parameters of the set of differential equations.
- Set new initial conditions *d(t*,s),* whenever a stochastic event occurs at *t**.
- Allow the deterministic process to switch between different dynamic behaviors, i.e. change of the vector functions that describe the system behavior f and g.

Algorithm

1. **Set initial states**

   The states of both the deterministic and the stochastic process are set at the initial time t_i ; $i = 0$.

2. **Evaluation of enabled set**

   Given the state of the system at t_i , i.e. $s(t_i)$ and (t_i) , the enabled transitions of the stochastic process are determined for t_i, as $act(s(t_i), d(t_i))$. A transition is enabled when the system occupies the state from which the transition is generated and the activation predicate *act* is true.

3. **Evaluation of next stochastic jump**

   The next transition and next jumping time t_{jump}, are identified through a "race-approach" in which only the enabled set of transitions participate. From those enabled transitions, the one with the minimum time to jump is selected. This approach is based on the parameter of transition *p(s,d).* Two sampling methods are proposed by Manno et al. (2015): event-driven and time-driven.

4. **Evaluation of the deterministic process until the time of the next stochastic jump**

   The deterministic process is evaluated until the next jumping time t_{jump} , identified in the previous step. The trajectory of the deterministic process can be evaluated by different approaches as numerical integration, closed form of analytical solution, etc.

   During this process, any changes that may occur in $act(s, d)$ (activation of stochastic transitions), need to be identified. In case of such change occurring, the evolution of the deterministic process is stopped at the time in which the change occurs t_e.

5. **Next simulation time.**

   The simulation time is updated after the evaluation of the trajectory of the deterministic process until t_{jump} . i.e. $= i + 1$, and the new $t_i \ = \min(t_{jump}, t_e)$.

6. **Evaluation of exit condition**

   The simulation time t_i is compared to the mission time of the analysis, $t_{mission}$. e.g. mission time could refer to the lifetime of the system under analysis. If $t_i > t_{mission}$ then the simulation goes to step 8, if not, to step 7.

7. **Update state**

The state of the system is updated at the current simulation time, t_i. From step 4, two situations are possible, either $t_i = t_{jump}$ or $t_i = t_e$, depending on the minimum between t_{jump} and t_e. In other words, either there is a change in the activation of stochastic transitions, or there is no change.

a) Change in enabled set. In this case, only the initial conditions of the deterministic process are updated, and no change in the stochastic process is done, i.e. $d(t_i) = d(t_{i-1})$, since the candidate jumping time t_{jump} from step 3, was not reached.

b) No change in enabled set. In this case, the candidate jumping time t_{jump}, from step 3 is accepted and the corresponding transition is fired. The stochastic process is then updated to the new state, and the deterministic model is updated consequently.

8. **Final state**

When the exit condition $t_i > t_{mission}$ is met, the state of the stochastic model at t_{i-1}, is registered as the final state at $t_{mission}$, and the state of the deterministic model is identified by the evaluation of the deterministic process until $t_{mission}$.

The discrete simulation algorithm flowchart is shown in Figure 2-7.

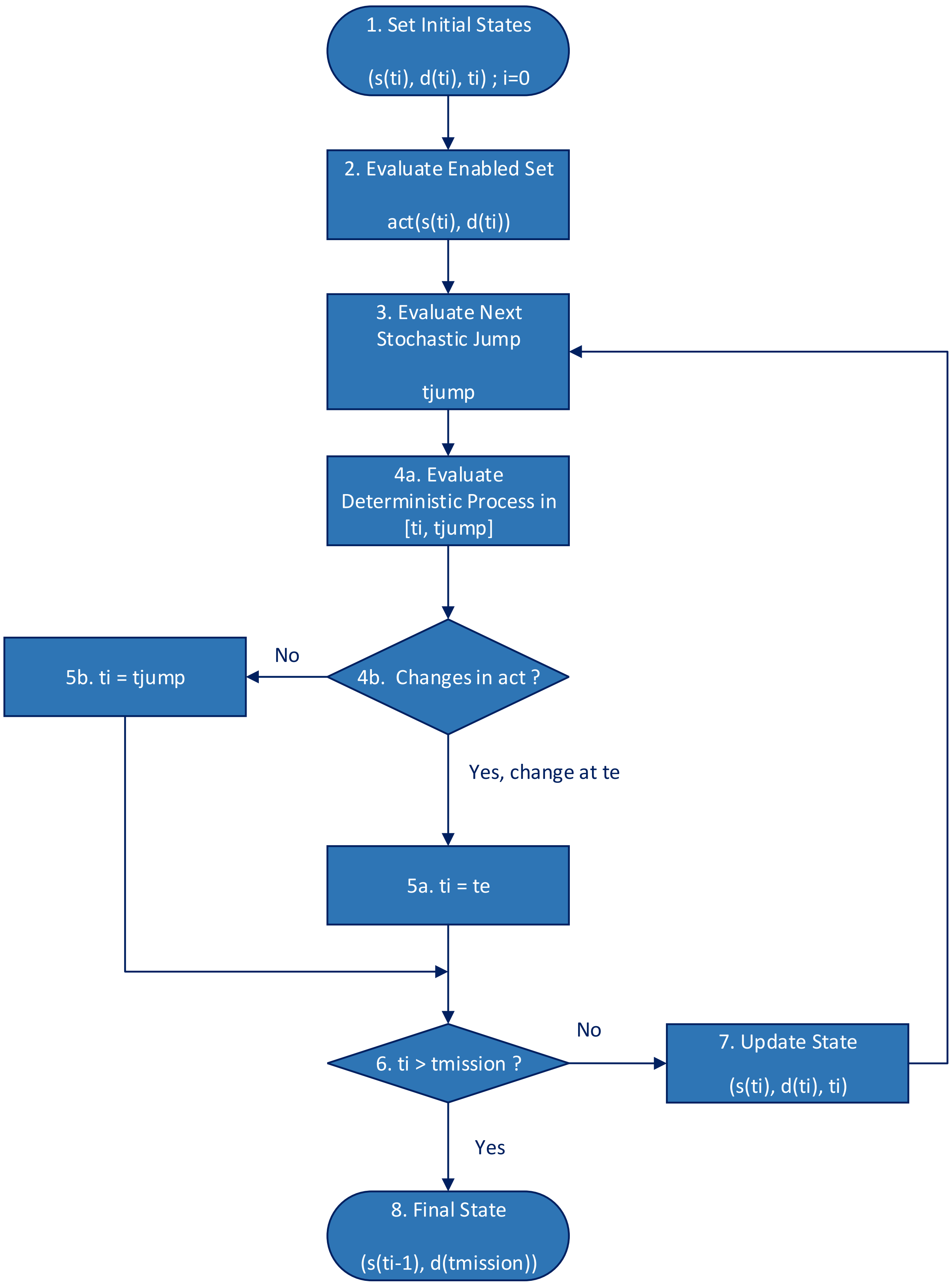


Figure 2-7. Discrete event simulation algorithm. (Adopted from Manno et al. (2013))

Sampling engines

For simulating the times to jump t_{jump} (step 3 of the discrete event simulation algorithm), two sampling methods are proposed by Manno et al. (2015): event driven and time driven.

Event driven sampling engine

The event driven sampling engine uses the thinning algorithm for simulating a Non-Homogeneous Poisson Process, a process in which the rate of occurrences of failures ($\lambda(t)$) varies with time.

The thinning algorithm consists of three main steps:

1. Defining an upper bound λ^* for $\lambda(t)$.
2. Sampling the candidate next jumping time t^*, using the inverse of an exponential distribution with rate λ^*.
3. Accepting the candidate sampled t^* , with probability $\lambda(t^*)/\lambda^*$

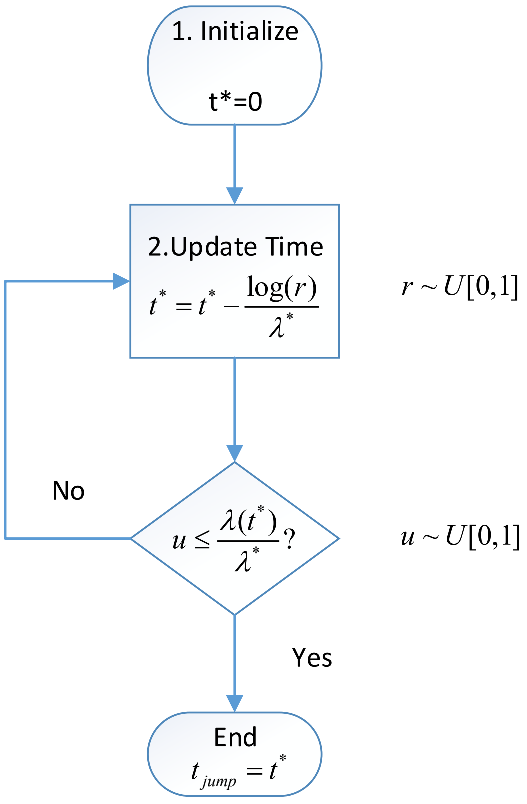


Figure 2-8. Thinning algorithm. (Adopted from Manno et al. (2015))

Figure 2-8, shows the thinning algorithm for simulating transition times with non-constant rate of occurrence. For the acceptance test, it is required to evaluate $\lambda(t^*)$, i.e. the rate of the transition at the candidate jumping time t^*.

Time driven sampling engine

The time driven sampling engine is based on step integration of the distribution function *F(t).* The probability that an item fails in an interval [0,t] can be obtained by the integration of the density function *f(t)*, as:

$$F(t) = Pr(T \le t) = \int_0^t f(u)du \;\; for\, t > 0 \tag{2-3}$$

And the probability density function *f(t),* is defined as:

$$f(t) = \frac{d}{dt}F(t) = \lim_{\Delta t \to 0} \frac{F(t+\Delta t) - F(t)}{\Delta t} = \lim_{\Delta t \to 0} \frac{Pr(t < T \le t + \Delta t)}{\Delta t} \tag{2-4}$$

So, when Δt is small, then:

$$Pr(t < T \le t + \Delta t) \approx f(t) \cdot \Delta t \tag{2-5}$$

This method then follows the evolution of the deterministic process with respect to time. At each integration step the relation of equation (2-5) is evaluated for those transitions of the stochastic process with parameters that are function of the deterministic process, $p = p(s, d)$, and compared to uniformly sampled random number between [0,1].

Hence, the main steps in the time driven algorithm are:

1. Define a small Δt .
2. Update the distribution function for the transitions that are dependent on the deterministic process, with the relation shown on equation (2-5), (step integration).
3. Accepting the candidate sampled t^* , with probability $f(t) \cdot \Delta t$.

The flowchart of the time driven sampling engine, is shown in Figure 2-9.

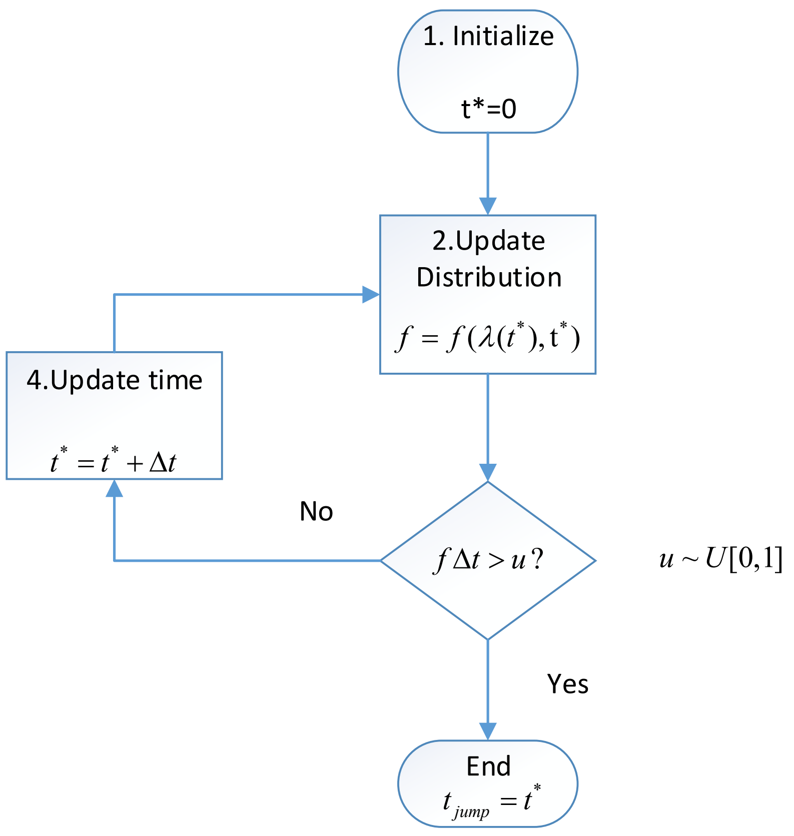


Figure 2-9. Time driven sampling algorithm. (Adopted from Manno et al. (2015))

2.5.2 Simplified discrete event simulation

The simplified discrete event simulation for dynamic reliability problems, is based on the discrete event simulation approach proposed by Manno et al. (2013). This simplified algorithm is applicable for hybrid models in which the stochastic process is not dependent on the evolution of the deterministic process (i.e. hybrid models with stochastic process of the discrete type and a soft coupling in the stochastic to deterministic direction).

Two major simplifications result from the stochastic process not being dependent of the deterministic:

1. Evaluating the next jumping time t_{jump}, (step 3 of the discrete event algorithm), can be done directly from the inverse distribution and sampling engines of the event-driven and time-driven type are not required. This simple fact can reduce the computational power required for the simulations and hence reduce simulation times significantly.
2. The steps 2, 4 and 5 of the proposed discrete event simulation algorithm can be skipped, resulting in more reduced computational power and simulation times.

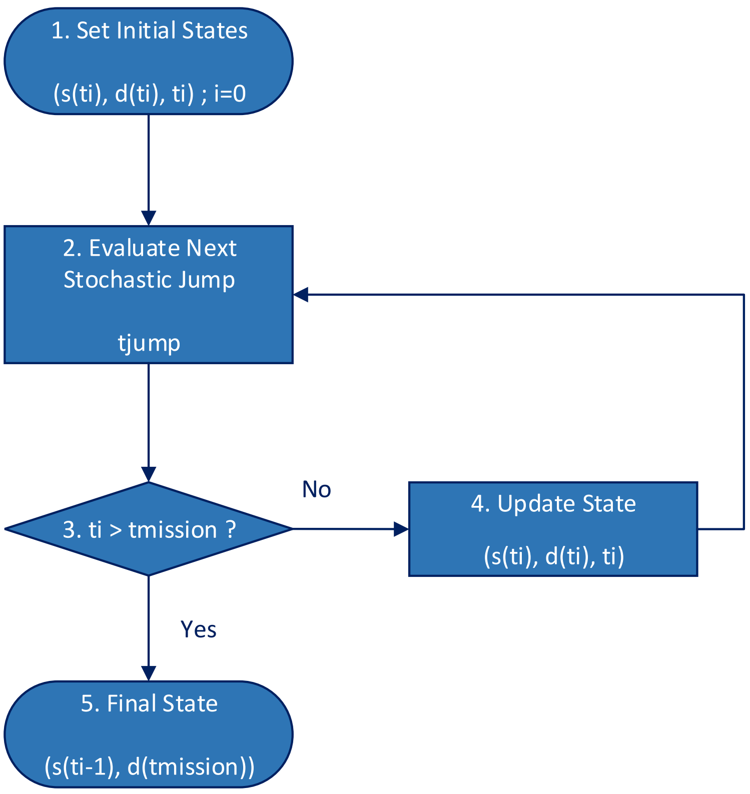


Figure 2-10. Simplified discrete event simulation algorithm.

Figure 2-10 shows the proposed simplification of the discrete event simulation algorithm.

2.5.3 Continuous simulation

From the classification proposed in Table 2-2 and the choice of simulation in Figure 2-6, continuous simulation can be applied for types of dynamical reliability problems in which both the deterministic process and the stochastic evolve in continuous form. This type of problems are not considered hybrid systems or hybrid stochastic system from the generalized definitions in which hybrid systems correspond to models with continuous and discrete phenomena. However, this type of problems are of relevance to dynamic reliability and give the opportunity of coupling continuous degradation processes with the dynamical operational and environmental conditions of the system, as well as dynamic interactions between different components, either in tight or soft coupled models. Degradation processes are of special interest for condition monitoring for prognosis. Coupling continuous degradation of components with the systems dynamics can provide better understanding of the degradation process and its influencing factors, pointing into relevant indicators for condition monitoring. The effect of degradation processes in components on the overall dynamic system can be observed.

This type of problems can be solved through simple continuous simulation. Both the deterministic and stochastic process evolve continuously over time and continuous simulation is a technique to solve the set of resulting stochastic differential equations numerically.

2.6 Dynamic reliability estimation

The problem of calculating the dynamic reliability of a system, i.e. the ability/probability of the system to perform its function considering dynamic operational and environmental conditions for a stated time period, can be approached through by formulating it as a reachability problem and Monte Carlo approximation.

In the analysis of hybrid systems, as pointed by Kowalewski (2002), the most important algorithmic verification procedure is reachability analysis, because many problems can be reduced to a reachability problem. A reachability analysis in hybrid systems answers the question of whether a certain hybrid state (composed of a discrete state and a region in the continuous space) is reachable from the initial hybrid state, in other words, if there is a path leading from the initial hybrid state to another hybrid state.

In the safety and reliability analysis context, reachability analysis are of interest to find not only whether an undesirable state (defined by stochastic and/or deterministic variables) is reachable but also what is the likelihood of this occurring. In this sense, failures of the system are identified, simulations of the dynamic model are carried out checking the reachability of these failure states, and the dynamic reliability of the system can be approximated by running multiple executions of the simulations (Monte Carlo method). Figure 2-11 illustrates the dynamic reliability formulated as a reachability problem. The paths marked in red represent executions in which the system failure state is reached.

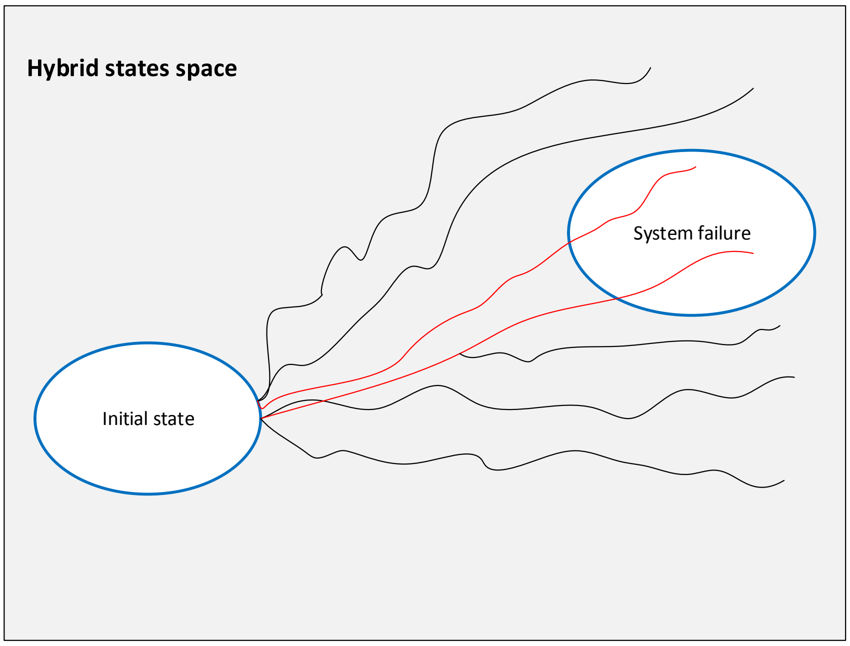


Figure 2-11. Dynamic reliability simulations as reachability problem.

2.6.1 Monte Carlo method

The Monte Carlo method can be used to approximate the dynamic reliability of a system in the following way:

1. Simulate N times with executions of $time\colon [0, t]$. Using either the discrete event simulation, simplified discrete event simulation algorithm or continuous simulation, as required.

2. Count the number of simulations that reach a system failure state.

$$Reach(i) = \begin{cases} 1 \; if \; system \; failure \; state \; is \; reach \\ 0 \; otherwise \end{cases} \tag{2-6}$$

3. Approximate the dynamic reliability $R_d(t)$ of the system as:

$$\hat{R}_d(t) = \frac{1}{N} \sum_{i=1}^{N} Reach\,(i) \tag{2-7}$$

$$\lim_{N \to \infty} \hat{R}_d(t) = R_d(t) \tag{2-8}$$

With the purpose of identifying, selecting and modelling a relevant subsea separator case study, the following chapters present the process description of the system to study, the degradation process in components involved in the system and the modelling of these processes, based on literature review.

Chapter 3

Subsea separation systems and models

3.1 Subsea processing

In the oil and gas industry, the term subsea is used to relate to the exploration, drilling, development, production and processing of oil and gas fields located underwater.

Subsea processing refers to the active handling of fluids that is performed at the seabed. Systems for subsea processing are becoming more common as solutions to enhance field economics. Subsea processing systems as pointed out by FMC-Technologies (2017) can enable solutions that are cost-effective and environmentally friendly without platforms for new applications (Greenfields), or can allow to extend the life of existing applications (Brownfields) while contributing to increase recovery and production.

Some of the main drivers for subsea processing systems as indicated by FMC-Technologies (2017) are:

- Increased recovery and field life extension.
- Production acceleration.
- CAPEX reduction on topside processing equipment and piping.
- Providing options for development of challenging fields (long distance tie-backs, reservoirs with low pressure, low permeability).
- Reduced CO_2 footprint compared to topside processing systems.
- Elimination of hazards and risks from severe weather conditions (cyclones, hurricanes, others).

Subsea processing includes system for separation, pumping, compression, heating or cooling. Currently, a key research area for subsea developments focuses on separation with the aim of establishing new separation concepts and equipment that are capable to operate over long times without the need of frequent interventions (SUBPRO, 2017).

3.2 Subsea oil/water separation

One of the major separation requirements in crude oil treatment is to recover the crude oil from its mixture with produced water. This water removal process is often carried out in oil/water separators working by sedimentation principle. Performing the water removal at the seabed allows injecting the water back to the reservoir and presents advantages over separating topside.

As presented by Statoil (2016), some of the drivers for subsea water removal and injection are:

- Increased production rate with pressure support in the reservoir.
- Less handling of water needed topside.
- Improved flow management (reduced pressure drop in flow lines, less slugging, reduction or elimination of the need of hydrate inhibitors).
- Improved HSE by less human exposure.

At the present, there are three oil/water separators installed subsea for water removal and injection. Table 3-1 shows the subsea separation projects installed and their equipment type, as presented in Statoil (2016).

Table 3-1. Subsea oil/water separators.

Name	Year of installation	Separation equipment	Pumps	Operator / Supplier	Water depth
Troll Pilot	1999	Horizontal gravity separator	Water injection pump	Statoil / GE	340
Tordis	2007	Compact horizontal gravity separator	Water injection pump and multiphase booster	Statoil / FMC	200
Marlim	2011	Pipe separator and hydrocyclones for removal of sand and polishing of water	Water injection pump	Petrobras / FMC	800

As Das et al. (2017) point out, gravity separators are simple and yet effective under flow conditions that change rapidly, providing volume for slug handling. Gravity separators have been extensively used in topside applications and now are used in subsea applications.

3.3 Gravity separator

Arntzen (2001) has reported that in oil/water gravity separators three layers can be distinguished: oil, emulsion and water. This three layers principle has been used as base to develop a dynamic model of a gravity separator by Das et al. (2017). A schematic representation of a three-phase gravity separator commonly used for water removal applications in the oil and gas industry is shown in Figure 3-1.

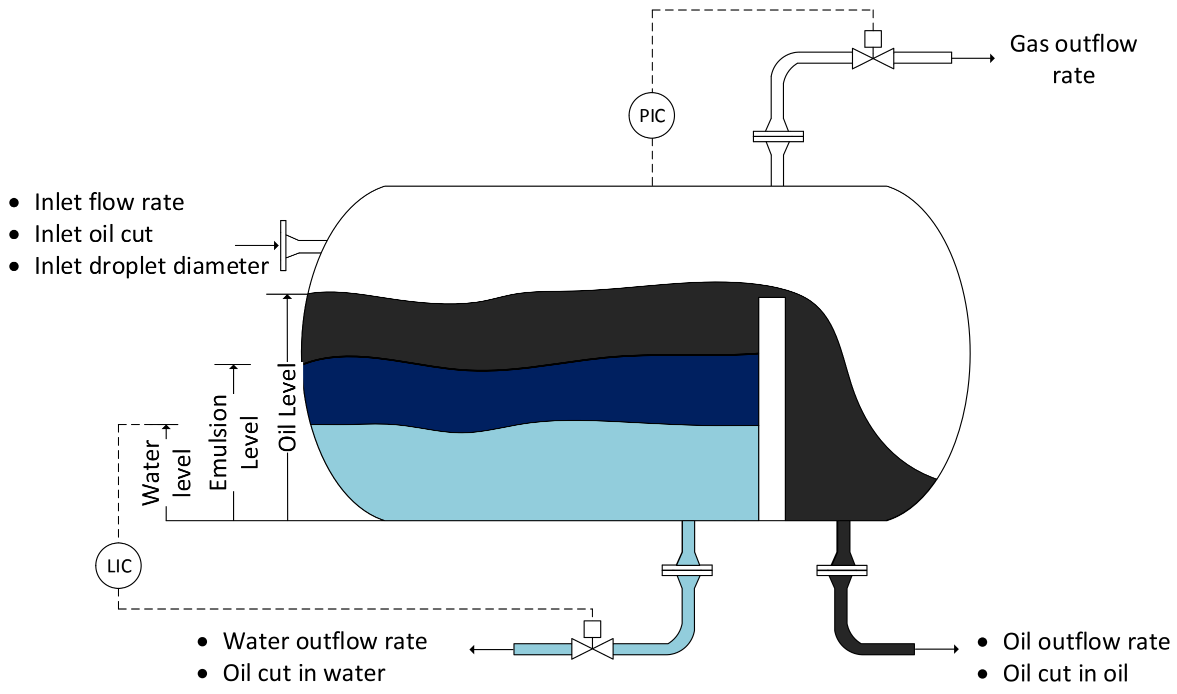


Figure 3-1. Gravity separation schematic. (Adopted from Das et al. (2017))

In this process, as presented by Das et al. (2017), a mixture of hydrocarbons and water consisting of oil and gas from the upstream wells flows into the separator. Most of the gas content is flashed out immediately with only minor traces remaining in the liquid layers at the outlet.

The liquid is separated into three distinct layers due to the differences in densities: an oil layer at the top, an emulsion layer at the middle and a water layer at the bottom.

A weir plate is placed in the separator so the oil layer is able to flow above it while the water layer is contained. In this sense, the oil level is self-regulated by the flow to the oil outlet zone. The oil flows out from the oil outlet nozzle for downstream process.

The emulsion layer located at the middle may grow or shrink in size depending on the separation extend.

The water flows from the bottom of the separator through the water outlet nozzle and the flowrate is regulated in order to keep the water level at a desired set point. The pressure inside the separator is regulated by the gas outflow.

The gas pressure and the liquid level inside the separator have to be measured for control. To measure the level of the three layers of liquid inside the separator the technology usually uses measurement of densities.

3.3.1 Gravity separator model

The dynamic model for the gravity separator developed by Das et al. (2017) makes the following assumptions:

- The inlet stream to the separator is a water in oil emulsion that develops into two interfaces: a sedimentation interface and a coalescence interface.
- Uniform droplet size is assumed, considering that in a continuous separation process, sedimentation and coalescence occur fast and the effect of distribution of varying droplet sizes can be neglected.

Figure 3-2 shows the separation process as modelled by Das et al. (2017). In the figure, the volumes for each of the liquid layers are shown. Each liquid layer volume contains its own oil cut represented by ε with the corresponding layer subscript.

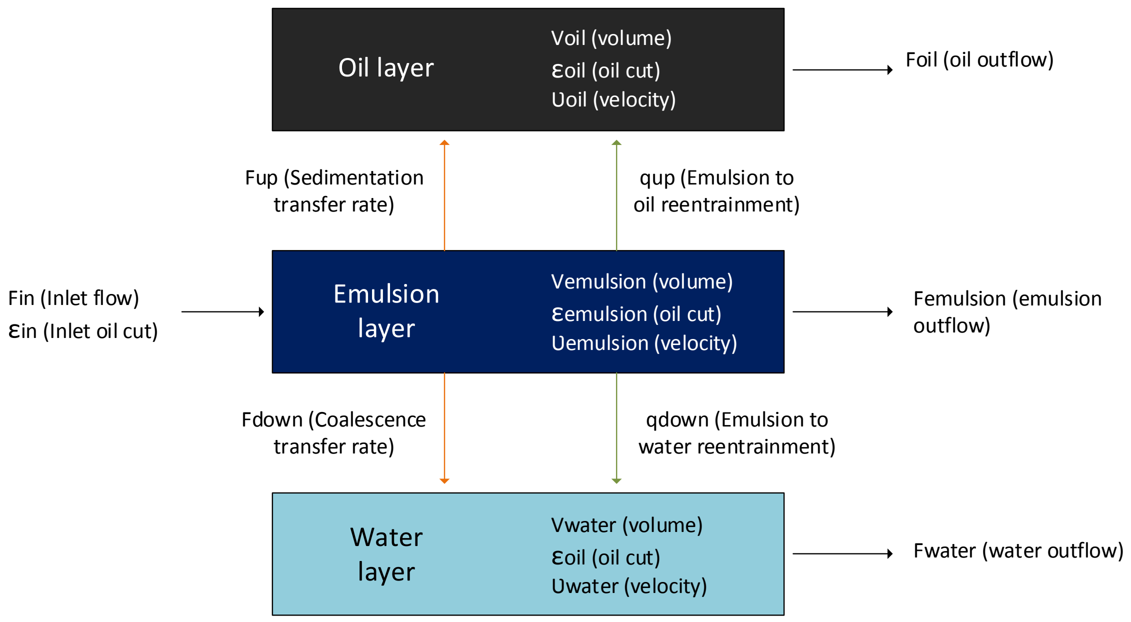


Figure 3-2. Flows and variables of the model (Adopted from Das et al. (2017))

F_{in} denotes the inlet flow, which contains the oil cut ε_{in} and goes directly into the emulsion layer. The emulsion layer, as mentioned in the assumptions, has sedimentation interface at the top and coalescence interface at the bottom.

F_{up} denotes a pure stream of oil flowing from the emulsion layer to the oil layer and F_{down} a pure stream of water flowing from the emulsion layer to the water layer. The oil and water layer are contaminated by turbulence and shear between the layers so the layers are not pure homogeneous. These contaminations are accounted by q_{up} and q_{down}, which are derived heuristically and represent streams of re-entrainment of emulsion to oil and emulsion to water, respectively.

The outlet flows are denoted: F_{oil}, representing the flow of oil over the weir plate and out the oil outlet nozzle, $F_{emulsion}$, representing a possible outflow of emulsion if the emulsion level is higher than the weir plate, and F_{water}, the outflow of water which corresponds to the manipulated variable for regulating the water level inside the separator.

Das et al. (2017) derived the model for the separator by using six volume balances: three total volume balances for the level of each layer and three oil volume balances for the oil cut in each layer.

The total volume balances that derive the level (height) of each layer, denoted h_{water}, $h_{emulsion}$ and h_{oil}, are described by the following equations:

$$\frac{\partial h_{water}}{\partial t} = \frac{F_{down} + q_{down} - F_{water}}{2L\sqrt{2rh_{water} - h_{water}^2}} \tag{3-1}$$

$$\frac{\partial h_{emulsion}}{\partial t} = \frac{F_{in} - F_{up} - q_{up} - F_{water} - F_{emulsion}}{2L\sqrt{2rh_{emulsion} - h_{emulsion}^2}} \tag{3-2}$$

$$\frac{\partial h_{oil}}{\partial t} = \frac{F_{in} - F_{oil} - F_{emulsion} - F_{water}}{2L\sqrt{2rh_{oil} - h_{oil}^2}} \tag{3-3}$$

The oil volume balances that derive the oil cut in each layer, denoted ε_{water}, $\varepsilon_{emulsion}$ and ε_{oil}, are described by the following equations:

$$\frac{\partial \varepsilon_{water}}{\partial t} = \frac{\varepsilon_{emulsion}.\, q_{down} - \varepsilon_{water}.\, F_{water} - \varepsilon_{water}.\, L.\, \frac{\partial A_{c_{water}}}{\partial t}}{L.\, A_{c_{water}}} \tag{3-4}$$

$$\frac{\partial \varepsilon_{emulsion}}{\partial t} = \frac{\varepsilon_{in}.\, F_{in} - F_{up} - \varepsilon_{emulsion}.\, F_{emulsion} - \varepsilon_{emulsion}.\, L.\left(\frac{\partial A_{c_{emulsion}}}{\partial t} - \frac{\partial A_{c_{water}}}{\partial t}\right) - \varepsilon_{emulsion}.\left(q_{up} + q_{down}\right)}{L.\left(A_{c_{emulsion}} - A_{c_{water}}\right)} \tag{3-5}$$

$$\frac{\partial \varepsilon_{oil}}{\partial t} = \frac{F_{up} + \varepsilon_{emulsion}.\, q_{up} - \varepsilon_{oil}.\, F_{oil} - \varepsilon_{oil}.\, L.\left(\frac{\partial A_{c_{oil}}}{\partial t} - \frac{\partial A_{c_{emulsion}}}{\partial t}\right)}{L.\left(A_{c_{oil}} - A_{c_{emulsion}}\right)} \tag{3-6}$$

In equations (3-4) to (3-16) the notation A_c is used to refer to the cross-sectional area of a layer or weir plate while notation A_t is used to refer to the transfer area of a layer, as shown in Figure 3-3. As shown in the figure, L corresponds to the effective length of the separator where the separation occurs and r corresponds to the radius of the separator.

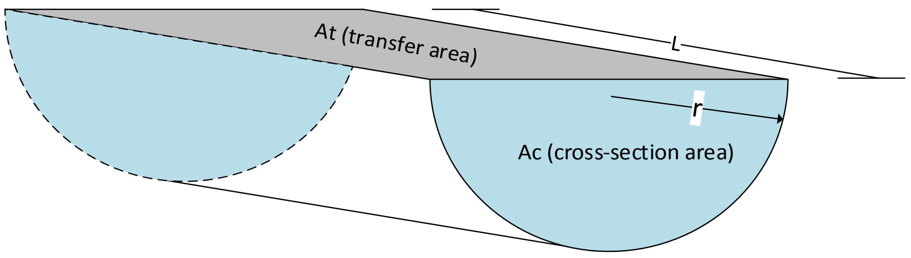


Figure 3-3. Areas used in the volume balance equations. (Adopted from Das et al. (2017))

The transport equations for F_{down}, F_{up}, q_{down},q_{up}, as described by Das et al. (2017) are:

$$F_{down} = P_{F_{down}} \cdot \left[\frac{2}{3}\frac{d}{t_{coalescence}} \cdot A_{t_{water}}\right] \tag{3-7}$$

$$F_{up} = P_{F_{up}} \cdot A_{t_{emulsion}} \cdot \frac{\mu_{oil}.Re_s}{\rho_{oil}.d} \tag{3-8}$$

$$q_{down} = P_{q_{down}} \cdot max\left(0\,;\, A_{t_{water}}.v_{water} \cdot \frac{h_{emulsion} - h_{water}}{h_{oil}}\right) \tag{3-9}$$

$$q_{up} = P_{q_{up}} \cdot max\left(0\,;\, k.A_{t_{emulsion}}.v_{oil} \cdot \frac{h_{emulsion} - h_{water}}{h_{oil}}\right) \tag{3-10}$$

$$k = min\left(\sqrt{\frac{h_{emulsion}}{h_{weir}}}\,;\, exp\left[-100 \cdot \left(\frac{h_{emulsion}}{h_{weir}} - 1\right)\right]\right) \tag{3-11}$$

Where: $P_{F_{down}}$, $P_{F_{up}}$, $P_{q_{down}}$, $P_{q_{up}}$ are tuning parameters; d is the inlet droplet diameter; $t_{coalescence}$ is the coalescence time; v_{water} and v_{oil} are the horizontal velocities of the water and oil stream through the separator; μ_{oil} is the oil viscosity; ρ_{oil} is the oil density and Re_s is the sedimentation Reynold's number and the procedure to compute it is described by Henschke et al. (2002).

The horizontal velocities of the water and oil streams are:

$$v_{water} = \frac{F_{water}}{A_{c_{water}}} \tag{3-12}$$

$$v_{oil} = \frac{F_{oil}}{A_{c_{oil}} - max\left(A_{c_{emulsion}} - A_{c_{weir}}\right)} \tag{3-13}$$

The total outflow over the weir plate can be calculated by Bernoulli equation as:

$$F_{out} = \frac{2}{3}C_d \cdot l_{weir} \cdot [max(0;\, h_{oil} - h_{weir})]^{1.5} \cdot \sqrt{2g} \tag{3-14}$$

Where C_d is a discharge constant, l_{weir} is the length of the weir, h_{oil} and h_{weir} are the height of the oil layer and height of the weir plate respectively and g is the acceleration due to gravity.

The oil outflow is then calculated as:

$$F_{oil} = F_{out} \cdot min\left(1\,; max\left(0\,; \frac{A_{c_{oil}} - A_{c_{emulsion}}}{A_{c_{oil}} - A_{c_{weir}}}\right)\right) \tag{3-15}$$

The possible emulsion outflow (i.e. In case of the emulsion level being higher than the weir plate) is the difference of the total outflow and the oil outflow.

$$F_{emulsion} = F_{out} - F_{oil} \tag{3-16}$$

When a separator has been designed, built and installed, the levels inside the separator are the available variables to control the behavior of the unit. They allow to control the velocities and residence times for the different layers in the separator, ensure the internals are operate under the environment they have been designed to work in and ensure that the downstream process is fed with fluids that are suitable for their requirement (Arntzen, 2016).

3.4 Level control system

As pointed out by Campos et al. (2010) one of the most important control in industrial systems is level control. A level control system is responsible for the mass balances of industrial facilities. In order to keep a constant level inside a recipient of constant volume, it is needed that the inlet mass flow rate is equal to the outlet mass flow rate.

As mentioned in section 3.3.1, in the gravity separator the water outflow rate F_{water}, is regulated in order to keep the water level at a desired set point.

The control system of the gravity separator is shown in Figure 3-4. The system operates in a closed-loop mechanism. The measured variable h_{water} is compared to the desired set point and the error between them is used by the controller to generate a control command U_c. The control command U_c drives the actuator of the valve that regulates the water outflow F_{water}.

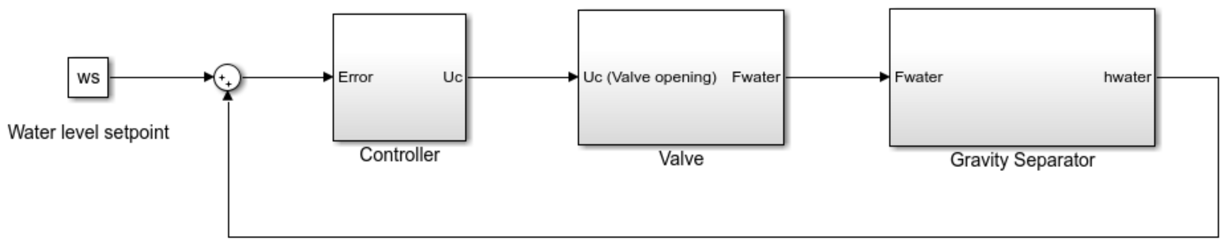


Figure 3-4. Schematic of the gravity separator control system.

3.4.1 Control valve

A control valve is an automated valve that can make adjustments in order to regulate the flow through a piping system. They are considered the final element in the control loop.

As described in the standard API-RP-553 (2012) and shown in Figure 3-5, a control valve consist of two major subassemblies:

1. A valve body.
2. An actuator.

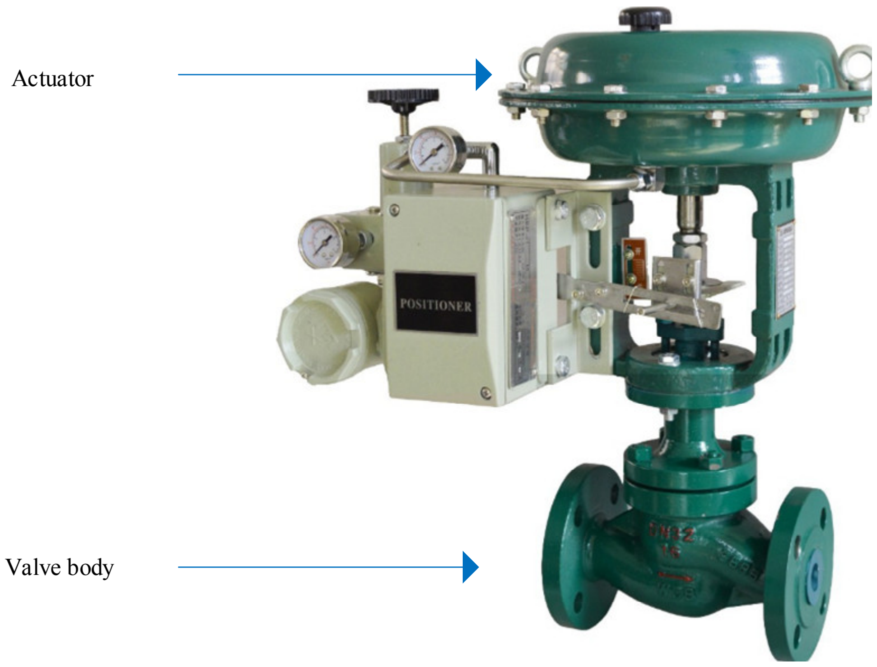


Figure 3-5. Control valve subassemblies.

The valve body is the part of the valve that actually contains the process fluid. It consists of a body, internal trim, bonnet and in some cases a bottom and/or bonnet flange. This subassembly must meet the requirements of the user’s design specification related to applicable pressure, temperature and corrosion, as stated in API-RP-553 (2012). The body of a control valve can be of different types such as: gate, plug, globe, butterfly, ball and others. Table 3-2 shows general characteristics as well as advantages and disadvantages of common valve types, as shown in whatispiping.com (2016).

Operating a control valve involves moving its movable part (plug, valve or vane) in relation to the stationary seat. The actuator is the subassembly in charge of physically positioning the valve’s movable part in response to the command from the automatic control system or from a manual device. The actuator must overcome the forces acting within the body subassembly by applying the adequate thrust while at the same time it must be responsive enough in order to position the valve plug accurately during changing process demands. Typical types of actuators include: manual, pneumatic, hydraulic and electric.

As shown in Table 3-2, globe valves present many advantages over other types of valves and this makes them the most common valve body style used as a control valve as mentioned in whatispiping.com (2016), providing an effective mean to regulate and control flow.

Table 3-2. Characteristics of different valve types. (whatispiping.com, 2016)

Type of valve					
	Gate	Plug / Ball	Globe	Butterfly	Diaphragm
Type of service	On / Off (sliding)	On / Off (rotary)	Throttling	Throttling	Throttling
Sealing method	Gate face slides parallel to the seat surface. Gate and seal in constant shear contact.	Radial seal, shaped to conform with ball surface	Disk motion perpendicular to valve seat. Only contact is in fully closed position.	Throttle blade is mashed into mated seal.	Diaphragm material is forced onto valve seat. Only contact is in fully closed position.
Advantages	Virtually no pressure loss across the valve face. Can be used for fluid with suspended solids.	Similar properties to gate valves. Lightweight, compact design. High capacity. Good range ability. Tight shutoff.	Good sealing characteristics. Can be used in frequently demanded process. Quick change of trim without removing valve from line. High capacity. Good range ability. Low noise trim available. Smooth control.	Lightweight, compact design. Minimal pressure loss across valve face. Low cost. High throughput capacity. Smaller shaft and actuator.	Almost no leakage, process fluid is isolated from valve stem. Self-cleaning.
Disadvantages	Poor sealing characteristics	Poor sealing lability with metal seats at high temperatures. Limited temperature range with resilient seats. Choke flow problems. Cavitation problems. Requires removal for maintenance.	High-pressure losses due to contorted path through the valve. Low noise trim reduces capacity.	Poor sealing characteristics. Good control limited to 60 deg - opening. Tight shut-off requires special lining plus over-sized shaft and actuators. Lining imposes temperature limitations.	Limited operating pressure. Limited temperature. High wear and tear. Poor control over 50% opening.

3.4.2 Globe valve

A globe valve is a valve with a linear or rotary closure element and a body that is distinguished by a globular shaped cavity around the port. A globe valve can have one or more ports. Figure 3-6. Parts of a globe style valve.Figure 3-6 shows the typical parts of a globe valve.

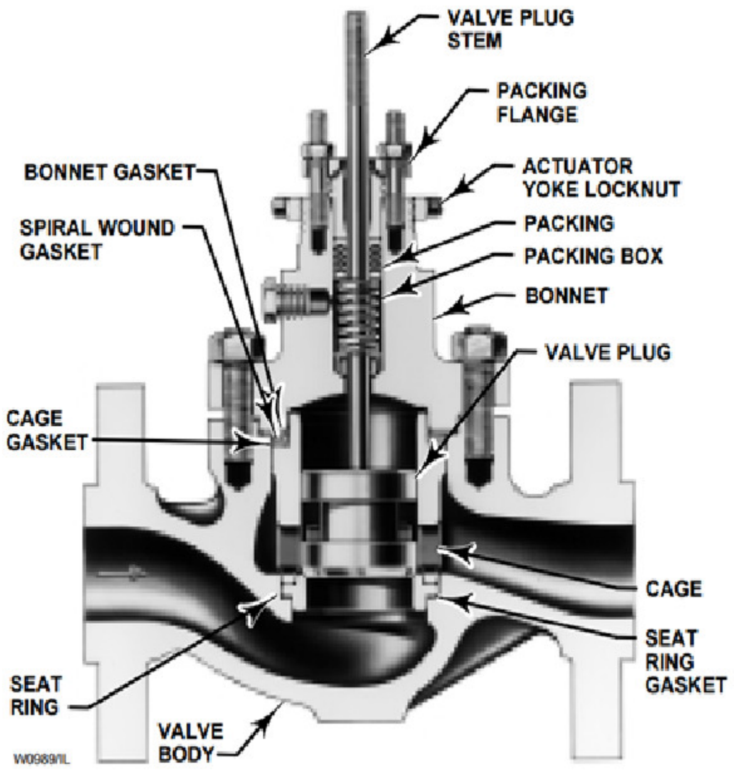


Figure 3-6. Parts of a globe style valve.

3.4.3 Valve inherent flow characteristic

The valve inherent characteristic is defined in API-RP-553 (2012) as “the relationship between the flow coefficient and the closure member (plug, ball, disk) as it is moved from the close position to rated travel with constant pressure drop across the valve”.

This characteristic is usually plotted as shown in Figure 3-7, where the horizontal axis represents the percent travel while the vertical axis represents the percent flow. The valve flow is a function of both the valve travel and the pressure drop across the valve. Flow characteristics tests are conducted at a constant pressure drop as a way to compare one valve characteristic to another. The typical valve flow characteristics are: quick opening, linear and equal percentage as shown in Figure 3-7.

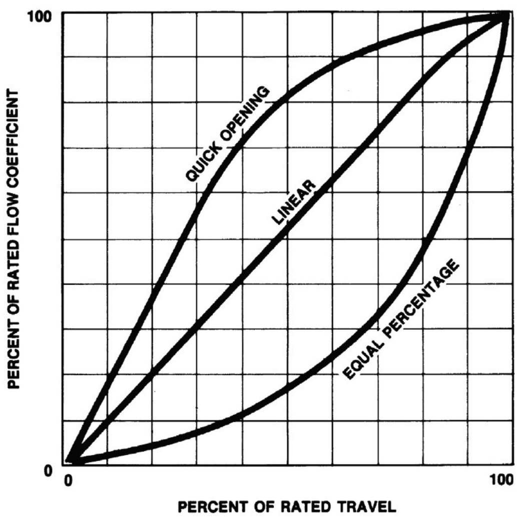


Figure 3-7. Inherent valve characteristics (API-RP-553, 2012).

The flow characteristic of a valve is achieved by different plug and/or cage designs, as shown in Figure 3-8.

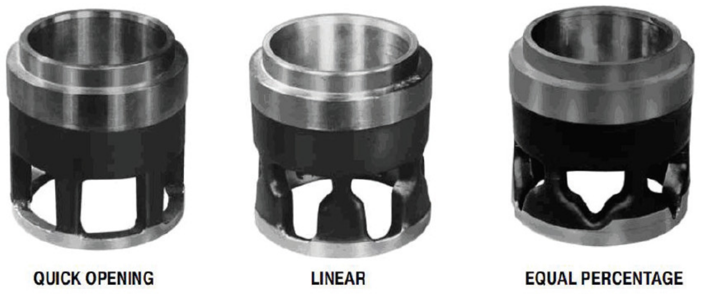


Figure 3-8. Cages for globe-styled valve bodies (API-RP-553, 2012)

As mentioned by Campos et al. (2010), in practice it is usual to approximate the dynamics between the signal of the control command, $U_c(t)$ and the position of the valve, $p(t)$ by a first order transfer function with dead-time, as:

$$p(s) = \frac{k_a \cdot e^{-T_d \cdot s}}{T_{63} \cdot s + 1} \times U_c(s) \tag{3-17}$$

Where k_a is the gain of the position with respect to the control command, T_d is the dead-time and T_{63} is a time constant that corresponds to the time for the valve to reach 63% of its final value after a change in its control command.

For a valve with a linear inherent flow characteristic, the flow is linearly proportional to its position with a gain K_v, so the flow is given by:

$$F(s) = K_v \cdot p(s) = \frac{K_v \cdot k_a \cdot e^{-T_d \cdot s}}{T_{63} s \cdot +1} \times U_c(s) \tag{3-18}$$

A control valve can be subject to degradation processes that may affect the control capabilities and thus the performance of the separation process. The next chapter presents the description of the degradation processes to which different subassemblies of the valve may be subject and the modelling of these processes, based on literature review.

Chapter 4

Degradation processes and models

4.1 Valve body erosion

Erosion is defined in Emerson (2011) as the weakening and gradual reduction of a valve’s body or trim due to severe process conditions such as dirty process, cavitation, flashing and outgassing. Figure 4-1

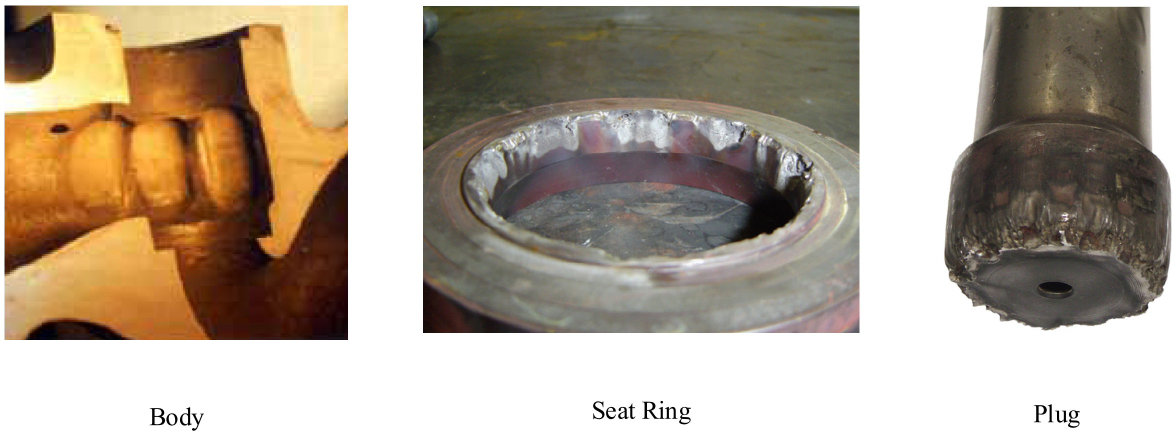


Figure 4-1. Erosion damages to valve parts.

Dirty process

Dirty processes can contain particulates that are abrasive to the valve's body and trim. In subsea applications, the production of sand is a known problem that can eat away internals of a valve.

Flashing

Flashing is the sudden evaporation of a liquid that occurs when the pressure of the liquid falls below its vapor pressure. The formed small vapor cavities can grind away material from the body of the valve and the trim.

Cavitation

Cavitation occurs when after flashing the pressure recovers to a value above the vapor pressure, causing the small vapor cavities to implode. It creates localized stresses inside the valve that can cause severe pitting to the body and trim.

Outgassing

Outgassing occurs when the pressure of a fluid falls below the saturation pressure of a dissolved gas in the fluid. As pointed by Emerson (2011), outgassing occurs very fast and is hard to predict. The gas is then separated from the solution producing erosive droplets that move at high velocities.

4.1.1 Flow coefficient as health indicator

The flow coefficient of a valve C_v, is defined as the number of U.S. gallons per minute of water that flows through a valve with a pressure drop of one pound per square inch. It is a constant related to the geometry of the valve, which can be used to establish the flow capacity for a given opening.

Erosion in the valve has an effect in its performance, such that when erosion occurs in the valve body or trim, the flow coefficient C_v increases due to increased valve opening.

$$D(t) = \Delta C_v = C_v^{act}(t) - C_v^{des} \tag{4-1}$$

As mentioned in previous work by Nystad et al. (2010) and Zhang et al. (2016), the difference between a theoretical designed C_v^{des} and a C_v^{act} that is calculated using the actual process parameters (i.e. differential pressure, flow rate, temperature, density), has been used as a useful indicator for valve erosion $D(t)$, as seen in equation (4-1).

4.2 Actuator loss of effectiveness

The ability of a control system to fulfill its intended function depends largely on its actuator. The mechanical and/or electrical parts of a valve actuator are subject to natural wear or ageing. As result of such wear, the effectiveness of the actuator, i.e. the ability to implement the control command U_c, is decreased and the performance of the control system is affected. This wear or ageing can be influenced by undesired effects of its operating conditions.

According to Langeron et al. (2015), much of the literature related to fault tolerant systems account for the loss of effectiveness of an actuator without information about the origin of the fault or its time to occur. The time for the occurrence of the fault is arbitrarily chosen and assumed to be deterministic, so the focus is placed on the re-allocation problem for the system to fulfil the control command U_c.

In their work, Langeron et al. (2015) propose the following modelling framework in which the loss of effectiveness of an actuator is explicitly linked to an underlying degradation process. It is assumed that:

- The more an actuator degrades, the more its effectiveness is decreased.
- A fault is a symptom of a reached degradation level, so the time of occurrence of the fault corresponds to the time to reach a degradation level.

The ability of the actuator to fully implement the control command $U_c(t)$ depends on the level of its physical degradation $D(t)$. The effectiveness of the actuator decreases from a value of 1 (the actuator works perfectly) to a lower limit $a \in [0,1]$, according to three different phases as shown in Figure 4-2.

- In Phase I, the actuator has full effectiveness even if it has some degradation, i.e. the effectiveness of the actuator is 1, while the degradation level is lower than a value b. The actuator robustness allows it to perform its function perfectly without being affected by its intrinsic damage.
- In Phase II, a linear decreasing of the effectiveness of the actuator occurs when the level of degradation is higher than b but lower than a value D_{max}. The actuator is sensitive to the degradation and performing its function has some difficulties.
- In Phase III, a saturation of the effectiveness of the actuator occurs. When the degradation level is higher than a given value of D_{max}. In this phase, the actuator continues to degrade and it is able to implement the control command in a reduced and constant way, independently of the degradation level.

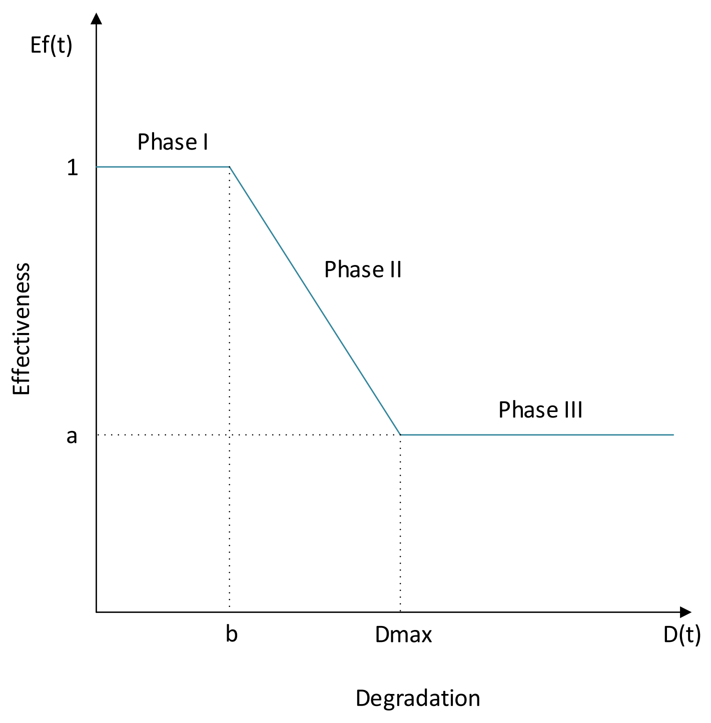


Figure 4-2. Loss of effectiveness of the actuator model (Langeron et al., 2012).

In this model, two faulty situations are included. The first one occurs at the time in which the degradation level reaches b and the second occurs at the time in which the degradation level reaches D_{max}. With these three phases different behaviors can be described with the parameters, for example, if $D_{max} = 0$, the effectiveness of the actuator decreases in linear form with the degradation until total failure of the actuator occurs, i.e. the actuator is not able to perform its function.

4.3 Modelling of degradation process

The degradation is considered a stochastic process, allowing to be independent of the technological features or materials of the valve's body, trim or actuator while taking uncertainties into account, as pointed by Langeron et al. (2015). Modelling the degradation as a stochastic process allows to account for uncertainties such as the sand production, the occurrence of flashing, cavitation, degassing and the conditions of the subsea environment in which the valve and its actuator operate.

The degradation is a monotone increasing process that can be modelled by different types of stochastic processes as Gamma, Exponential and others. As mentioned by Langeron et al. (2012), the choice depends on the type of the degrading component and/or the nature of the degradation.

Two types of model of the degradation process are presented. One model is not dependent on the evolution of the deterministic process or dynamic behavior of the process and the control law. This model can be used to construct a hybrid model of the coupling type with stochastic to deterministic direction of dependency, as presented in section 2.4.3. Another model, considers a degradation process of the actuator that is dependent on the evolution of the deterministic process variables, allowing to construct a tight-coupling hybrid model.

4.3.1 Degradation for soft-coupling hybrid model.

As pointed by Nguyen et al. (2015), the occurrence of partial loss of effectiveness of an actuator occurs as a discrete phenomenon in time. The same can be assumed for changes in the performance of the valve due to a degradation caused by sand, cavitation, flashing or outgassing that occur at discrete random times. In this sense, the occurrence of degradation is caused by a mechanism (shock) at given time intervals that leads to an increment in the accumulated damage. To model the degradation in this way, two random variables have to be included: the time of occurrence of shocks and the amount of damage caused by each shock.

It is considered that the instants of time at which wear amounts gradually accumulate, follow a homogeneous Poisson process (HPP) of intensity λ, as in the work by Nguyen et al. (2015). In a HPP, the times between occurrences of shocks are independent and exponentially distributed with parameter λ or analogically, the time to the j^{th} shock is Gamma distributed with probability density function:

$$f(t) = \frac{\lambda \cdot (\lambda t)^{j-1} \cdot e^{-\lambda t}}{(j-1)!} \tag{4-2}$$

Each time that a shock occurs, the component experiences an amount of damage W_j. This amount of damage is also a random variable that can be modelled with a Gamma, Normal, Uniform or other distribution. Then, the accumulated degradation at a time t is given by:

$$D(t) = \sum_{j=1}^{N(t)} W_j \; ; \; [N(t) = 0,1,2, \dots] \tag{4-3}$$

When modelling the degradation process this way, the possible impact of the control law on the degradation process is not taken into account, meaning that the hybrid model corresponds to a soft-coupling model in which the evolution of the physical deterministic process is impacted by the stochastic degradation process but not the other way around.

4.3.2 Degradation for tight-coupling hybrid model.

In their work, Langeron et al. (2012) propose a model of the degradation process of an actuator following the proportional hazard model (PHM) suggested by Cox (1972). In the model, the degradation speed is impacted by the control command $U_c(t)$. Using this model, $\Delta D(t)$, the increment of the degradation at time t, may be expressed as:

$$\Delta D(t) = \Delta D_0(t) \cdot e^{\beta_i} \tag{4-4}$$

Where $\Delta D_0(t)$ corresponds to the baseline degradation rate and β_i represents the effect due to the i_{th} state. The operational conditions of the actuator are summarized by Langeron et al. (2012) in seven states, defined by the factors $U_c(t)$ and $\frac{\partial U_c}{\partial t}$, under one single operational environment, as in Table 4-1.

Table 4-1. States of covariates. Cox model (Langeron et al., 2012)

State x_i	β_i	Comment
$\frac{\partial U_c}{\partial t} = 0\ ;\ U_c(t) \neq U_{max}, U_{min}$	β_1	Nominal condition
$U_c(t) = U_{max}$	β_2	High level saturation
$U_c(t) = U_{min}$	β_3	Low level saturation
$\frac{\partial U_c}{\partial t} > 0\ ;\ \frac{\partial U_c}{\partial t} < \xi$	β_4	Positive control variation
$\frac{\partial U_c}{\partial t} > 0\ ;\ \frac{\partial U_c}{\partial t} \geq \xi$	β_5	Possible damage
$\frac{\partial U_c}{\partial t} < 0\ ;\ \frac{\partial U_c}{\partial t} > -\xi$	β_6	Negative control variation
$\frac{\partial U_c}{\partial t} < 0\ ;\ \frac{\partial U_c}{\partial t} \leq -\xi$	β_7	Possible damage

In the table, the value ξ corresponds to a very high variation in the control command $\frac{\partial U_c}{\partial t}$, above which the control effort may have a relatively high impact on the degradation of the actuator. U_{max} and U_{min} are the physical saturations of the actuator. In a Cox model, there is no assumption about the form of the baseline degradation rate $\Delta D_0(t)$.

Therefore, the accumulated degradation at time t is:

$$D(t) = D(t-1) + \varDelta D(t) \tag{4-5}$$

When modelling the degradation process as a PHM, the degradation is a product of one factor $\Delta D_0(t)$ which is a function of time t, and another factor that is a function of the state x_i that depends on the control command U_c and its variation $\frac{\partial U_c}{\partial t}$. Taking into account the effect of the control command and its variations on the degradation process is relevant for systems in which the control variables (e.g. temperature, flow) have to track trajectories that vary with time according to production phases. These systems have a more dynamic operation with more frequent demands of the control system and the actuator, which may cause an impact on the degradation rate that should be considered as a tight-coupling hybrid model.

Chapter 5

Hybrid model of gravity separation system

When embarking in the task of modelling a dynamic system, the starting point should be to define the system to study and its boundaries. For this purpose, a functional and physical decomposition or breakdown of the system are recommended, not only facilitating the modelling procedure but also the analysis of results and the decision process that follows. This chapter presents the systematic procedure to build the hybrid model of the gravity separator case study with the selected tool.

Simulink® has been chosen as the tool for building the model and conducting the experiments. As described in MathWorks (2016), *"Simulink® is a block diagram environment for multi-domain simulation and Model-Based Design. It supports simulation, automatic code generation, and continuous test and verification of embedded systems".* Simulink® provides a graphical editor, block libraries that can be customizable, and solvers for the modelling and simulation of dynamic systems. It is integrated with MATLAB® allowing the user to export the simulation results to MATLAB® for further analysis, as well as incorporating algorithms from MATLAB® into the models. This integration with MATLAB® makes it a convenient tool for discrete event simulations and Monte Carlo simulations.

5.1 Functional and physical breakdown of the system

A functional breakdown or decomposition aims to provide a better understanding on how the overall function of a system is achieved by means of sub functions working together. It is the process of breaking down the overall function of the system into smaller parts.

The gravity separation system under study can be decomposed according to its function as shown in Table 5-1.

Table 5-1. Functional decomposition of the system.

Functions
1. Recover the crude oil from its mixture with produced water, according to design specifications.
1.1. Contain the production flow in a physically appropriate mean to facilitate the separation process.
1.2. Regulate the separation process to maintain performance specifications.
1.2.1. Regulate the interface level (water/emulsion) in the separator to guarantee the required transfer areas and residence time.
1.2.1.1. Measure the interface level inside the separator.
1.2.1.2. Compare the difference between the measured level and required set point and generate appropriate correction.
1.2.1.3. Regulate the water outflow to maintain the interface level as required.
1.2.1.3.1. Position the valve's plug according to the appropriate correction.
1.2.1.3.2. Restrict the pass of water to adjust its flow.

The sub-functions can be allocated to the sub-systems and components, from a physical decomposition or breakdown as shown in Figure 5-1

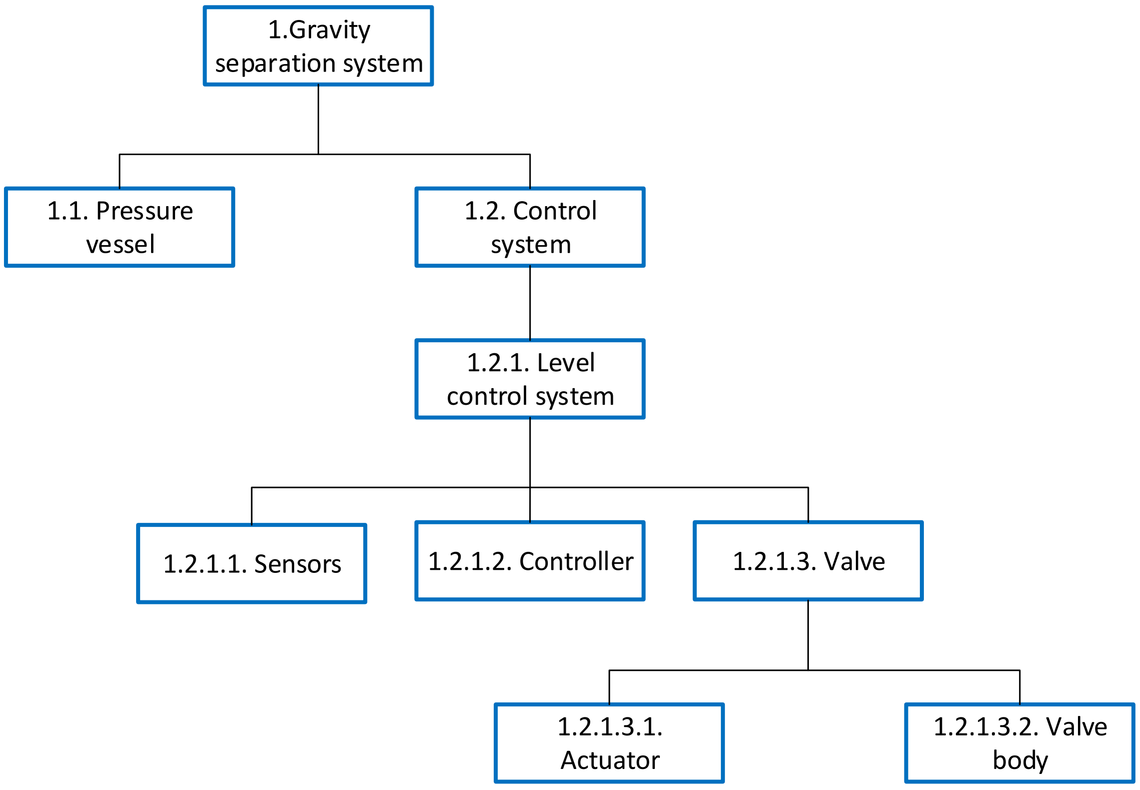


Figure 5-1. Physical decomposition of the system.

The hybrid model consists of the subsystems and components indicated in the physical decomposition and the sub-functions of the functional decomposition. Others such as a pressure control system are not included. In this sense, the hybrid model attempts to capture the dynamical behavior of the system shown, describing the function of the overall system in terms of deterministic and stochastic processes, according to parameters that describe the characteristics of the components and equations describing the behavior.

5.2 Deterministic model

The deterministic model refers to the process described in Chapter 3, including the equations derived from the balances in the gravity separator and the equations of the control law, assuming a linear inherent flow characteristic for the valve.

The parameters used for the gravity separator simulations are taken from the model by Das et al. (2017) and are shown in Table 5-2.

Table 5-2. Parameters for the gravity separator model. (Adopted from Das et al. (2017))

Separator dimensions	Tuning parameters in transfer equations	Fluid properties
$R = 2m$ $L = 10m$ $h_{weir} = 3m$ $l_{weir} = 3.5m$	$P_{F_{down}} = 1$ $P_{F_{up}} = 1$ $P_{q_{down}} = 0.5$ $P_{q_{up}} = 0.05$	$\rho_{water} = 1000\ {kg}/{m^3}$ $\rho_{oil} = 845\ {kg}/{m^3}$ $\mu_{water} = 0.5 \times 10^{-3}\ mPa.s$ $\mu_{oil} = 1.3 \times 10^{-3}\ mPa.s$
Inlet flow conditions	**Initial states**	**Others**
$F_{in} = 1.1\ {m^3}/{s}$ $\varepsilon_{in} = 0.55$ $d = 1.7\ mm$	$h_{water_0} = 1.56$ $h_{emulsion_0} = 1.67$ $h_{oil_0} = 3.21$ $\varepsilon_{water_0} = 0.1$ $\varepsilon_{emulsion_0} = 0.3$ $\varepsilon_{oil_0} = 0.9$	$t_{coalescence} = 0.12$ $g = 9.81\ {m}/{s^2}$ $C_d = {1}/{\sqrt{3}}$

The deterministic model is implemented in Simulink® using the appropriate blocks to introduce the required equations. An overview of the model is presented in Figure 5-2, showing the different subsystems involved. The level control system is as described in section 3.4. The control valve is split into two subsystems, actuator and valve (body), in order to allow modelling of a degradation process separately for each. In a non-degraded actuator $U_c = U_c^*$.

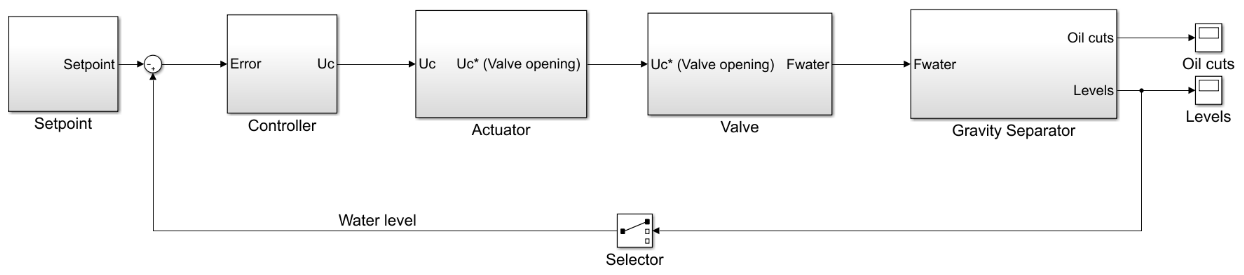


Figure 5-2. Overview of the gravity separator control system in Simulink®.

Figure 5-3 shows the model of the gravity separator as implemented in Simulink®. It includes the input parameters, the equations for calculating the transfer and cross-sectional areas of the different liquid layers in the separator, the transfer equations and the equations for the six states: liquid levels and oil cut in each liquid layer. These equations are presented in Chapter 3.

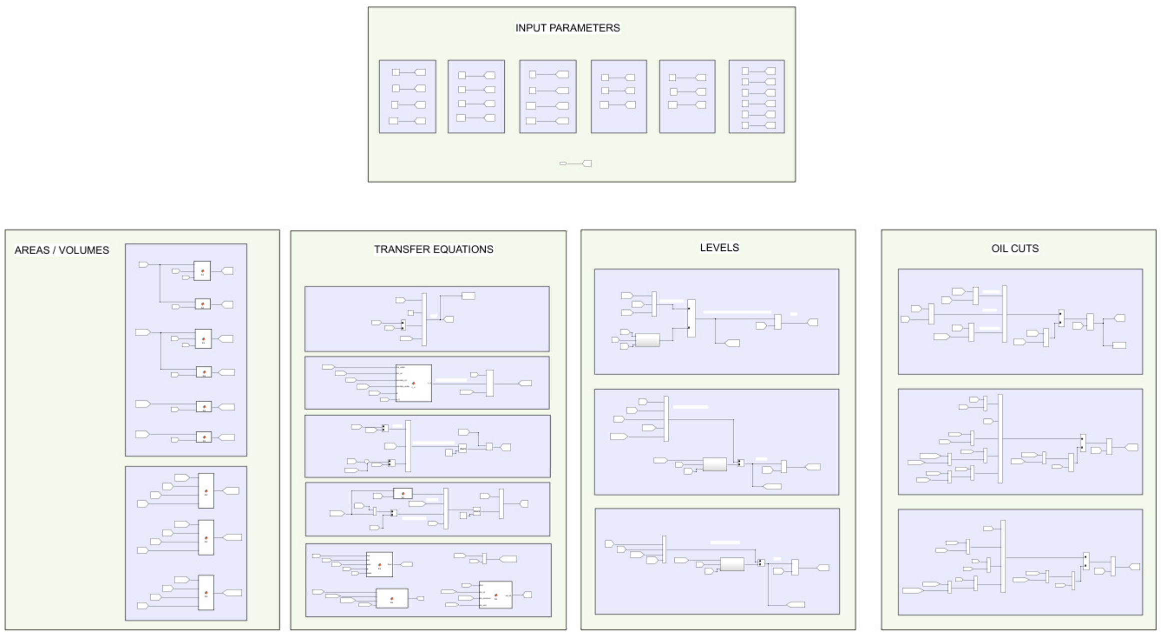
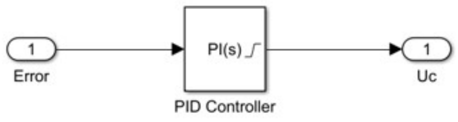
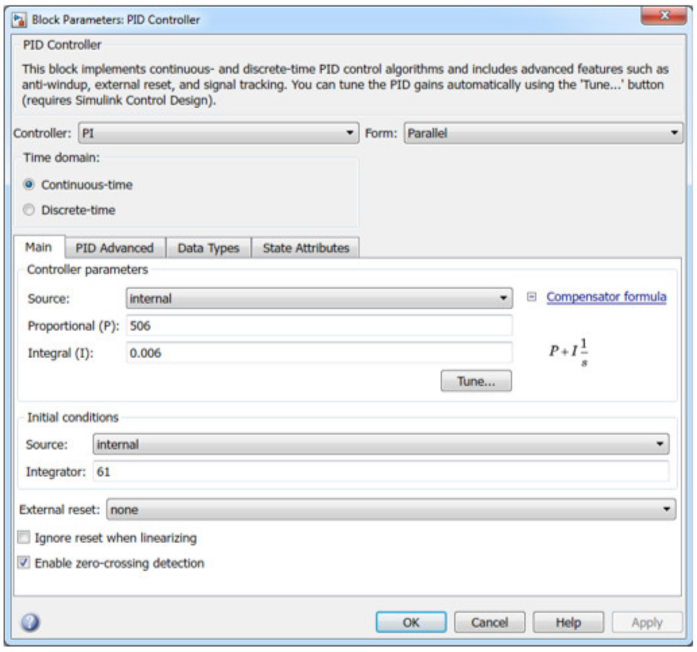


Figure 5-3. Overview of the gravity separator subsystem as implemented in Simulink®.

A controller of the type PI (Proportional-Integral) is placed in the model by using Simulink® built-in PID controller block. The controller is tuned using the “tune” function in Simulink®. The block and its parameters are shown in Figure 5-4.




Block Parameters: PID Controller

PID Controller

This block implements continuous- and discrete-time PID control algorithms and includes advanced features such as anti-windup, external reset, and signal tracking. You can tune the PID gains automatically using the 'Tune...' button (requires Simulink Control Design).

Controller: PI   Form: Parallel

Time domain:
[✓ Continuous-time]
Discrete-time

Main | PID Advanced | Data Types | State Attributes

Controller parameters
Source: internal
Proportional (P): 506
Integral (I): 0.006
Compensator formula: $P + I\frac{1}{s}$

Initial conditions
Source: internal
Integrator: 61

External reset: none

☐ Ignore reset when linearizing
☑ Enable zero-crossing detection

Figure 5-4. Simulink® controller configuration

5.3 Stochastic model

The stochastic model simulates the degradation process of the actuator or the body of the valve as presented in Chapter 4.

As presented in section 4.3, the degradation process is modelled as a compound Poisson process. In this process, the time between consecutive shocks is exponentially distributed with rate λ, and the amount of damage of each shock follows a uniform distribution between 0 and δ . The parameters used for simulations are shown in Table 5-3.

Table 5-3. Parameters for the degradation process

Poisson process intensity	Uniform distribution maximum (Max. amount of damage per shock)
$\lambda = 1 \times 10^{-3}\ \left({shocks}/{s}\right)$	$\delta = 0.005\ \left({damage}/{shock}\right)\ (max)$

The parameters for the model of the degradation process are significantly large in order to avoid high simulation times, i.e. a very fast degradation process is modelled to be able to couple with the deterministic process and run simulations in a reasonable time. Figure 5-5, shows several realizations of the degradation process. As will be discussed later in Chapter 6, a challenge with hybrid models is a difference in the time scales in which the deterministic process and the stochastic process evolve.

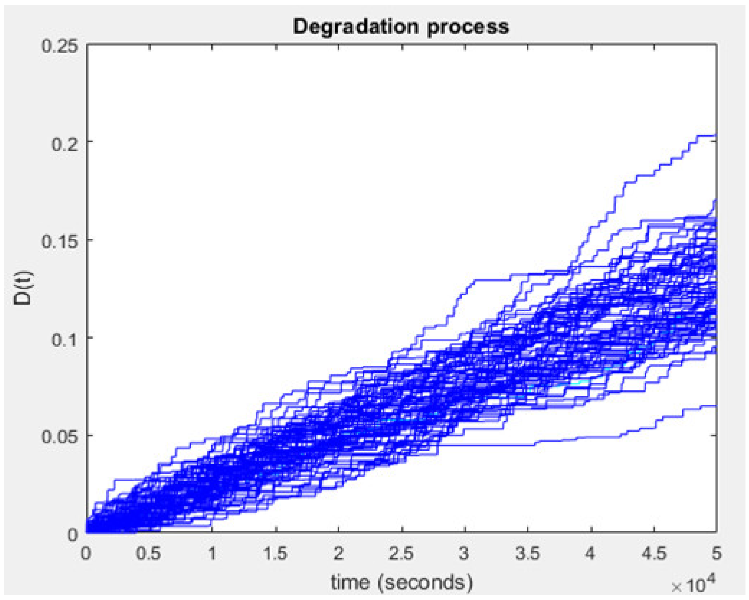


Figure 5-5. Stochastic degradation process.

Figure 5-6 shows that the degradation process can be coupled to the deterministic model via shared variables to the body of the valve or to the actuator.

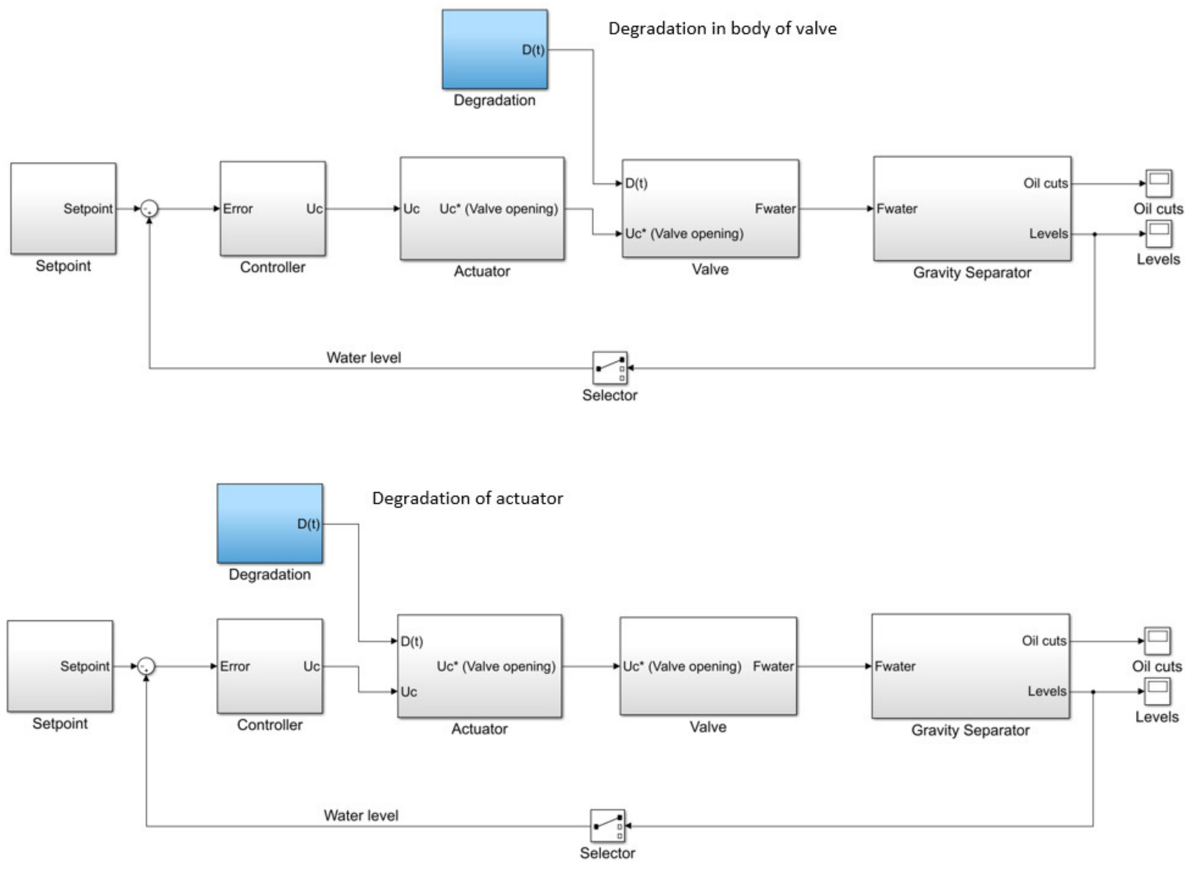


Figure 5-6. Overview of the model coupling degradation processes.

5.4 Coupling the models

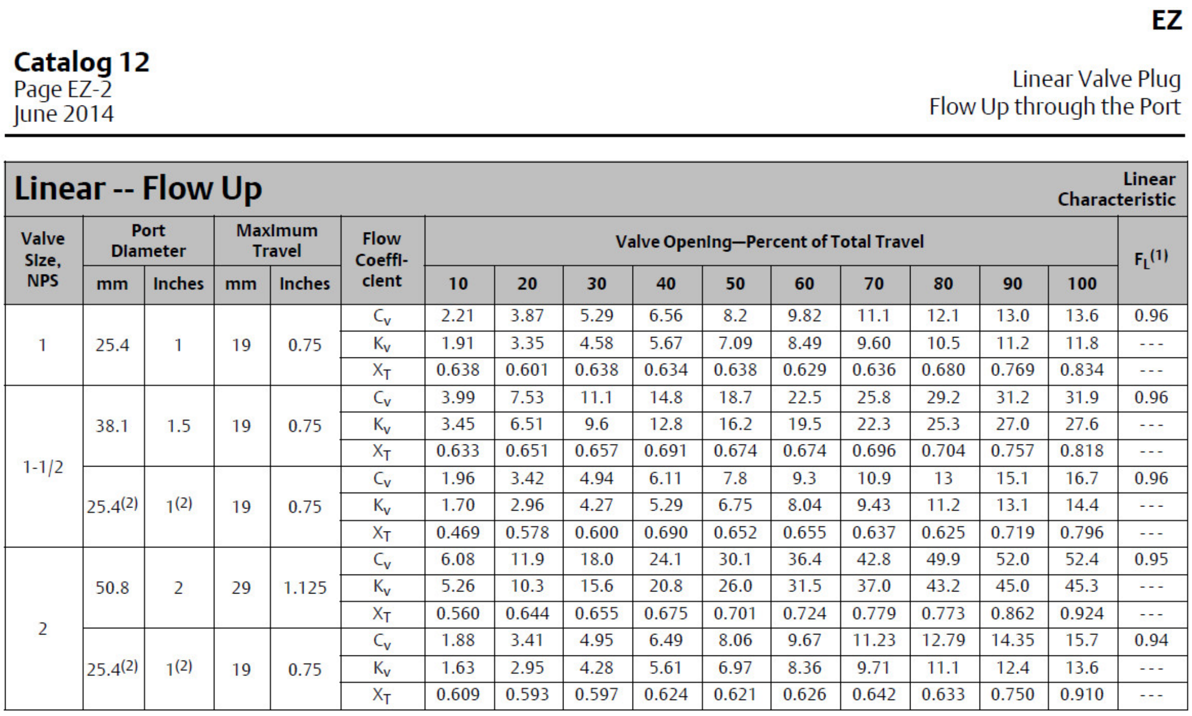
The deterministic and stochastic models are coupled by means of shared variables. Since the hybrid model of the case-study aims to analyze the performance of the system under degradation of the actuator of the valve or degradation in the valve body, the coupling shall be done in these components and its corresponding sub-functions as presented in the physical and functional decomposition of the system.

Under the assumption that the inlet flow conditions to the gravity separator are constant, as shown in Table 5-2, the system would operate on a fixed level set point, meaning that the control system would be on steady state for most part of the operation. The possible impact of the control law on the degradation process as consequence of the dynamics in the demand of the actuator or valve may be disregarded. It is then possible to build a soft-coupled model with stochastic to deterministic dependency, i.e. the evolution of the stochastic degradation process influences the development of the physical deterministic model, but not the other way around. This assumption simplifies the model and its simulation.

5.4.1 Valve body degradation

The function of the valve body, described in the functional decomposition as “1.2.1.3.2. Restrict the pass of water to adjust its flow” is affected by the degradation process to which the component is subject. In the deterministic model, the flow coefficient relates the opening of the valve (output from the controller U_c) with the flow across it, i.e. the water outflow rate F_{water}.

Part of a catalog from a control valve manufacturer is shown in Figure 5-7. A valve with a linear characteristic, the flow coefficient C_v and its equivalent metric flow factor K_v are shown.



EZ

Catalog 12
Page EZ-2
June 2014

Linear Valve Plug
Flow Up through the Port

Linear -- Flow Up

Linear Characteristic

Valve Size, NPS	Port Diameter		Maximum Travel		Flow Coefficient	Valve Opening—Percent of Total Travel										F_L(1)
	mm	Inches	mm	Inches		10	20	30	40	50	60	70	80	90	100	
1	25.4	1	19	0.75	C_v	2.21	3.87	5.29	6.56	8.2	9.82	11.1	12.1	13.0	13.6	0.96
					K_v	1.91	3.35	4.58	5.67	7.09	8.49	9.60	10.5	11.2	11.8	- - -
					X_T	0.638	0.601	0.638	0.634	0.638	0.629	0.636	0.680	0.769	0.834	- - -
1-1/2	38.1	1.5	19	0.75	C_v	3.99	7.53	11.1	14.8	18.7	22.5	25.8	29.2	31.2	31.9	0.96
					K_v	3.45	6.51	9.6	12.8	16.2	19.5	22.3	25.3	27.0	27.6	- - -
					X_T	0.633	0.651	0.657	0.691	0.674	0.674	0.696	0.704	0.757	0.818	- - -
	25.4(2)	1(2)	19	0.75	C_v	1.96	3.42	4.94	6.11	7.8	9.3	10.9	13	15.1	16.7	0.96
					K_v	1.70	2.96	4.27	5.29	6.75	8.04	9.43	11.2	13.1	14.4	- - -
					X_T	0.469	0.578	0.600	0.690	0.652	0.655	0.637	0.625	0.719	0.796	- - -
2	50.8	2	29	1.125	C_v	6.08	11.9	18.0	24.1	30.1	36.4	42.8	49.9	52.0	52.4	0.95
					K_v	5.26	10.3	15.6	20.8	26.0	31.5	37.0	43.2	45.0	45.3	- - -
					X_T	0.560	0.644	0.655	0.675	0.701	0.724	0.779	0.773	0.862	0.924	- - -
	25.4(2)	1(2)	19	0.75	C_v	1.88	3.41	4.95	6.49	8.06	9.67	11.23	12.79	14.35	15.7	0.94
					K_v	1.63	2.95	4.28	5.61	6.97	8.36	9.71	11.1	12.4	13.6	- - -
					X_T	0.609	0.593	0.597	0.624	0.621	0.626	0.642	0.633	0.750	0.910	- - -

Figure 5-7. Linear characteristic valve catalog (Emerson, 2014).

Since the process is modelled using the metric system, K_v is used instead of C_v. The equivalent metric flow factor K_v indicates the water flow in $\left({m^3}/{h}\right)$ at a pressure drop across the valve of 1 $\left({kg}/{cm^2}\right)$. This pressure drop is assumed for the model.

Degradation or erosion in the valve body has an effect in its performance causing the flow coefficient or the equivalent metric flow factor to increase as described in section 4.1.1. The flow factor K_v is then the shared variable for coupling the deterministic process of the control and the stochastic degradation of the valve body.

Figure 5-8 shows how the degradation process of the valve body increases the flow factor. While the valve degrades the flow factor is increased keeping its linear characteristic. A degraded valve may leak in fully closed position hence the flow factor is not zero at the fully closed position when the valve has some degradation.

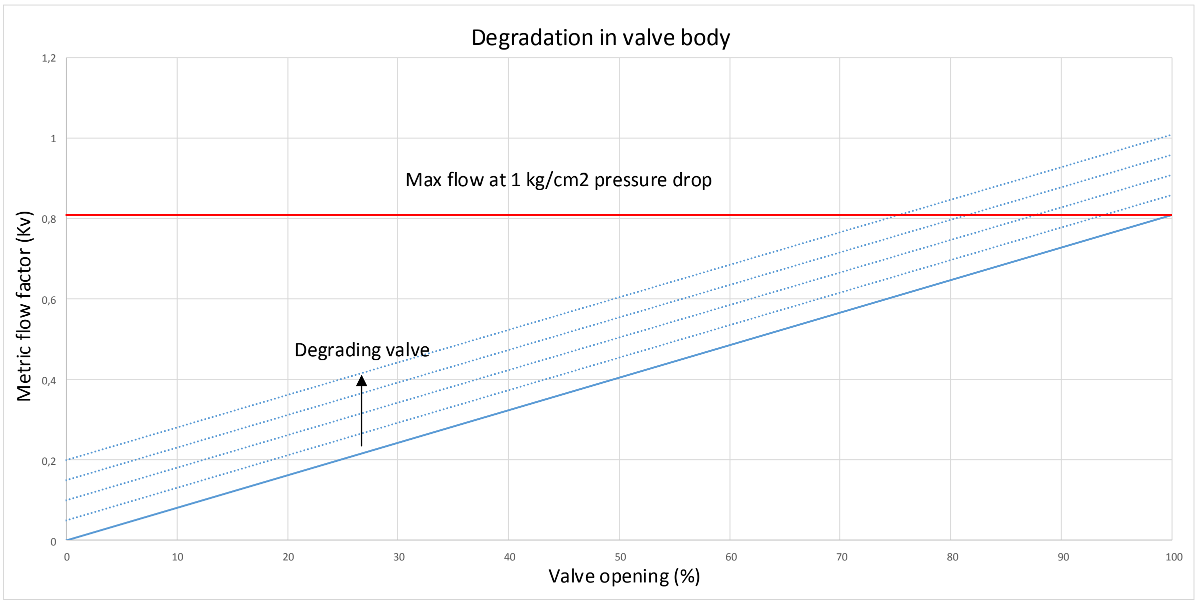


Figure 5-8. Degradation of valve body.

It is considered in the model that the maximum possible flow of the valve for a pressure drop of $1\ \left({}^{kg}/_{cm^2}\right)$, cannot be exceeded, i.e. when a valve degrades the flow factor is increased but a restriction in the maximum flow is placed. This consideration is valid under the assumption that the pressure drop across the valve remains at a constant value of $1\ \left({}^{kg}/_{cm^2}\right)$. Higher flow rates would be possible for higher pressure drops.

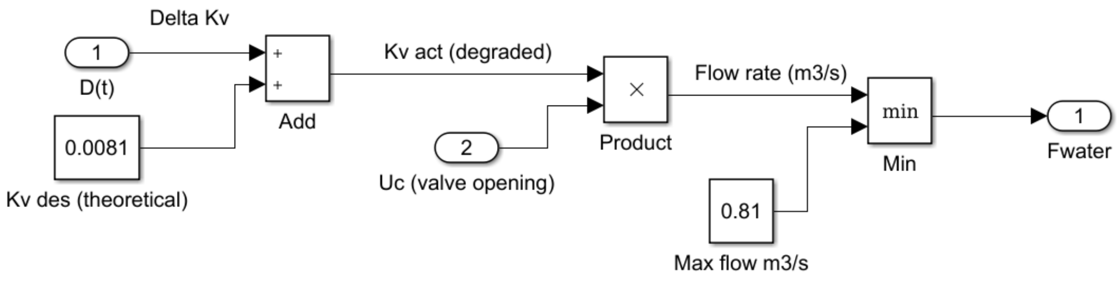


Figure 5-9. Coupling of degradation of valve body and control model in Simulink®.

Figure 5-9 shows the coupling of the deterministic and stochastic model for the degradation process in the valve body, as modelled in Simulink®.

5.4.2 Actuator degradation

A non-degraded actuator is fully capable of fulfilling its functions described in the functional decomposition as "1.2.1.3.1. Position the valve's plug according to appropriate correction", i.e. the actuator is perfectly capable of performing the control command U_c. As the actuator degrades, it experiences a loss in its effectiveness, the ability to perform the control command, in three different phases as presented in section 4.2. This means that the positioning of the valve is then a function of both the stochastic degradation process and the deterministic control law.

The positioning of the valve U_c^* is the shared variable to couple the deterministic and stochastic process when the degradation occurs in the actuator, as:

$$U_c^*(t) = Ef(t) \times U_c(t) \tag{5-1}$$

Where $Ef(t)$ is the effectiveness of the actuator, i.e. a function of the degradation process $D(t)$.

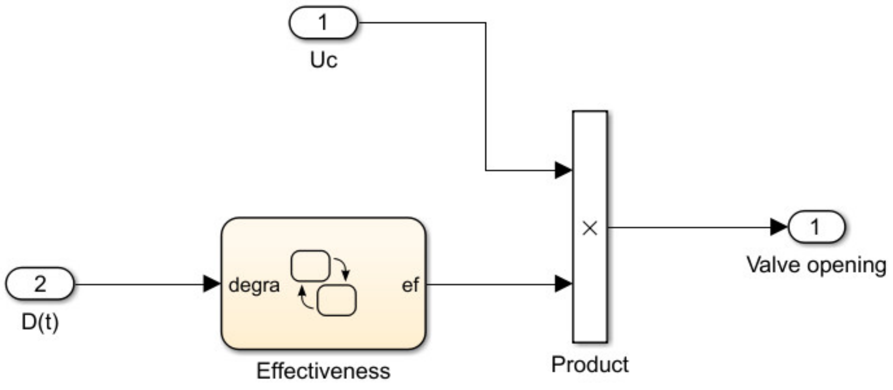


Figure 5-10. Coupling of degradation of the actuator and control model in Simulink®.

Figure 5-10 shows how the coupling of the stochastic degradation process of the actuator and the deterministic control law is done in Simulink®.

As presented in section 4.2, a degrading actuator experiences a loss of effectiveness according to three phases. In phase I, the actuator is perfectly able to perform the control command regardless its degradation. In phase II, the actuator effectiveness decreases as a lineal function of the degradation, and in phase III, the actuator effectiveness may saturate and keep a constant level after a degradation level is reached. For this model, the parameters considered for the effectiveness as function of the degradation are shown in Table 5-4. These parameters are determined heuristically, i.e. as a rule of thumb based on the degradation process model described in section 5.3. As mentioned, the parameters of the degradation process are significantly large to avoid large simulation times and hence are not supported by real data or experiments of degradation and loss of effectiveness process.

Table 5-4. Parameters for loss of effectiveness of the actuator.

Degradation level limit for Phase I	Degradation level limit for Phase II	Lowest effectiveness, i.e. at Phase III
$b = 0.032$	$D_{max} = 0.15$	$a = 0.4$

Figure 5-11 shows the development of one degradation process and the effectiveness of the actuator.

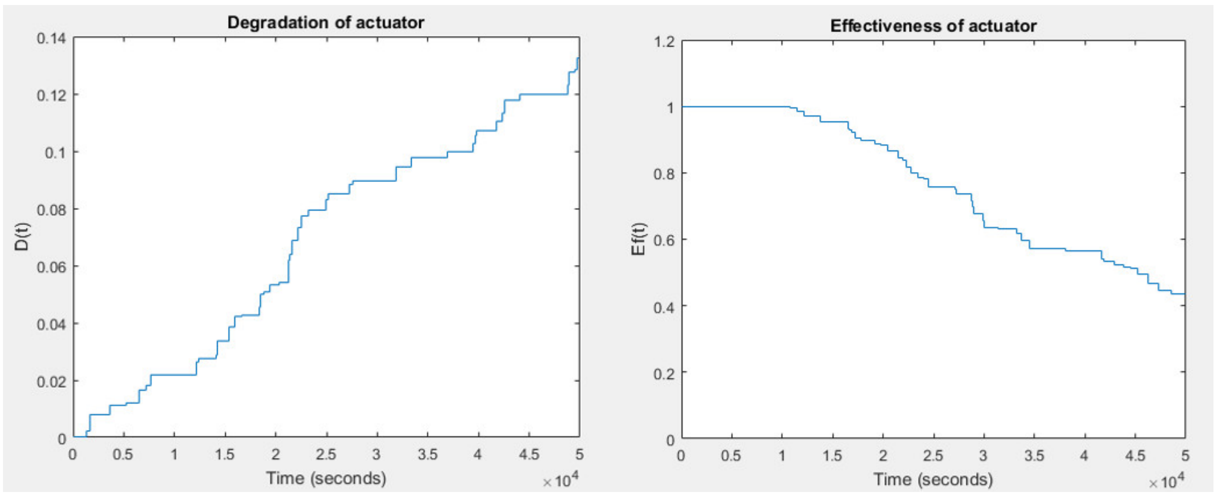


Figure 5-11. Degradation and effectiveness of actuator.

Chapter 6

Hybrid models application and challenges

This chapter elaborates on the potential of the application of hybrid models for dynamic reliability and performance assessments in the development of new technology for subsea processing facilities and presents discussions on the challenges of the application, using the case study of the gravity separator to demonstrate relevant points.

6.1 Application potential

With the goal of moving entire topside oil and gas production and processing facilities to the seabed, new technology developments are required. Regarding separation processes, research points towards the development of compact separator solutions with modular design to reduce the cost of oil/water subsea installations and hence make the business case for subsea water removal more attractive (SUBPRO, 2017). These compact solutions also require advanced control strategies that are more critical than in standard topside gravity separators because of the reduced buffer volume and difficulty to access. Moreover, as pointed by DNV-GL (2017), digitalization of the oil and gas sector is on the rise and is the area in which companies are more likely to invest across research and development, trials and full scale implementations during 2017. Digitalization is based on the use of instrumentation and dynamic models for advanced control and optimal remote operations.

A key research area in the RAMS field is driven by the challenge with the development of new technology, advanced control strategies, digitalization and the need to find optimal solutions with respect to safety, cost and performance. The aim is to introduce new models and communication platforms for the integration of reliability analysis with system design, at early stages, when key conceptual decisions are made (SUBPRO, 2017).

Hybrid modelling is an approach that can integrate dynamic reliability into the models used to digitalize the operation of different processes, allowing the development of new technology, such as compact separator solutions, while providing confidence of its safe and reliable operation, from early design stages. In this way, the design of the new technology for subsea applications can maintain optimal performance and continuous operations despite faults or failures in components of the system, through fault tolerant control.

6.1.1 Fault tolerant control

Fruitful results have been obtained from the research areas of fault tolerant control (FTC) and hybrid systems separately, however as mentioned by Yang et al. (2010) the problem of FTC of hybrid systems has not yet attracted enough attention and needs to be investigated both for academic and practical purposes. In automated processes, faults can cause undesired reactions and shutdown of the controlled plant causing damages to technical parts of the plant or to its environment, making fault diagnosis and FTC a requirement of complex control systems. The main idea of a fault tolerant control is to prevent a component fault from causing a failure at the system level.

A fault in this context is described as a change in characteristics of a component that affects its operation or performance in an undesired way. A failure is the inability of a component or system to perform its function. In this sense, a fault tolerant control aims to handle or “work around” faults in order to keep the faulty system operational (Blanke et al., 2016).

In the gravity separation case study, faults correspond to the loss of effectiveness in the actuator or erosion in the valve. A hybrid model of the control loop allows testing the design in order to determine the necessary measures to ensure fault tolerance.

Fault tolerant control is classified in two major subgroups: passive and active (Alwi et al., 2011) as shown in Figure 6-1.

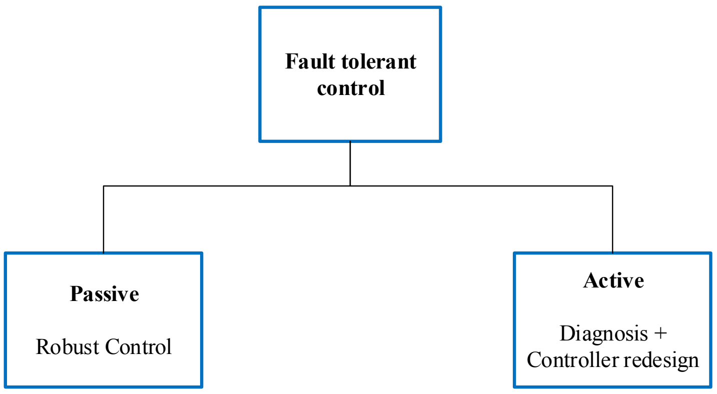


Figure 6-1. Classification of fault tolerant control

A passive FTC is designed in a way that the system is able to achieve its objectives without changes in the control law. Robust control is a method that provides passive fault tolerance. The controller is fixed and the system is designed to satisfy its goals in healthy situation and under the presence of faults.

An active FTC responds to faults in the system by reconfiguring the control law in order to maintain an acceptable performance of the system. They require a supervision level to perform diagnosis providing the required information about the fault so that the control law can be reconfigured accordingly, as shown in Figure 6-2.

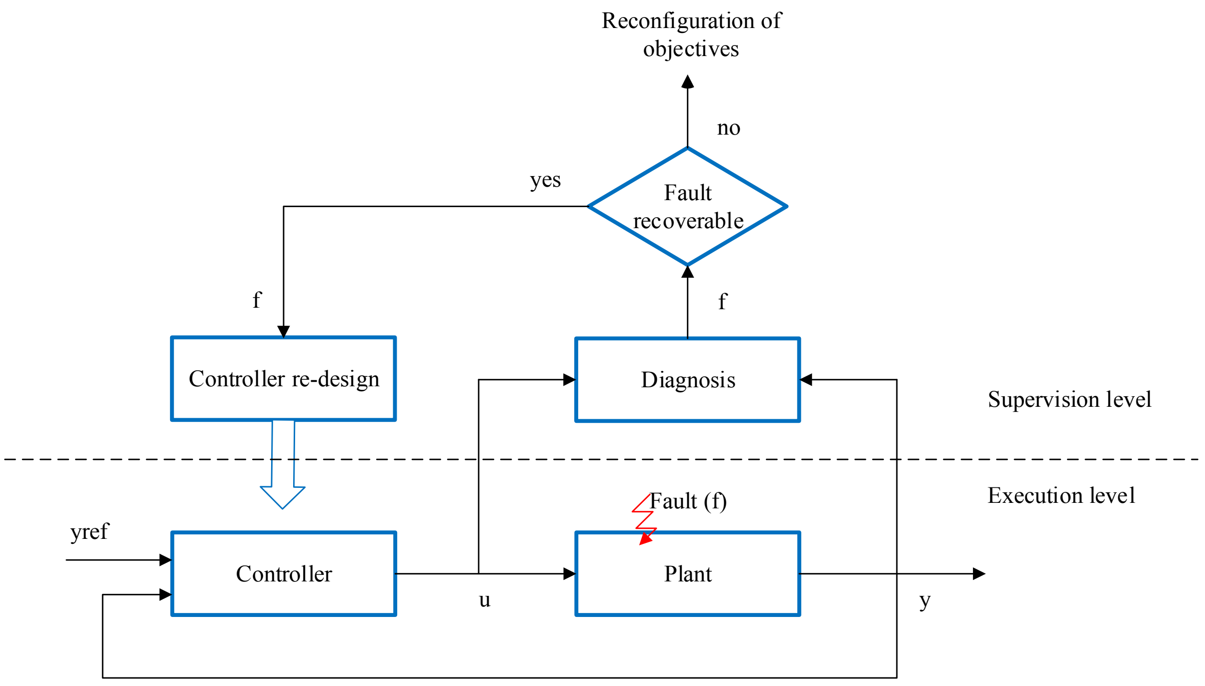


Figure 6-2. Architecture of active fault tolerant control. (Adopted from Blanke et al. (2016))

With the goal of developing new technology, digitalization of processes, with advanced fault tolerant control strategies, and the need to find optimal solutions with respect to safety, cost and performance, technology qualification provides a systematic approach and frame that can be applied from early stages of the development. A technology qualification process is a useful recommended practice to ensure that new technology will function within the optimal performance limits with acceptable confidence.

6.1.2 Technology qualification program

An illustration of the technology qualification program as presented in the systematic approach in DNV-RP-A203 (2011) is shown in Figure 6-3 (a). The program iterates through three stages: concept evaluation, pre-engineering and detailed engineering. In each of these three stages, a cycle of the technology qualification process (b) is to be concluded before moving onto the next stage of the development.

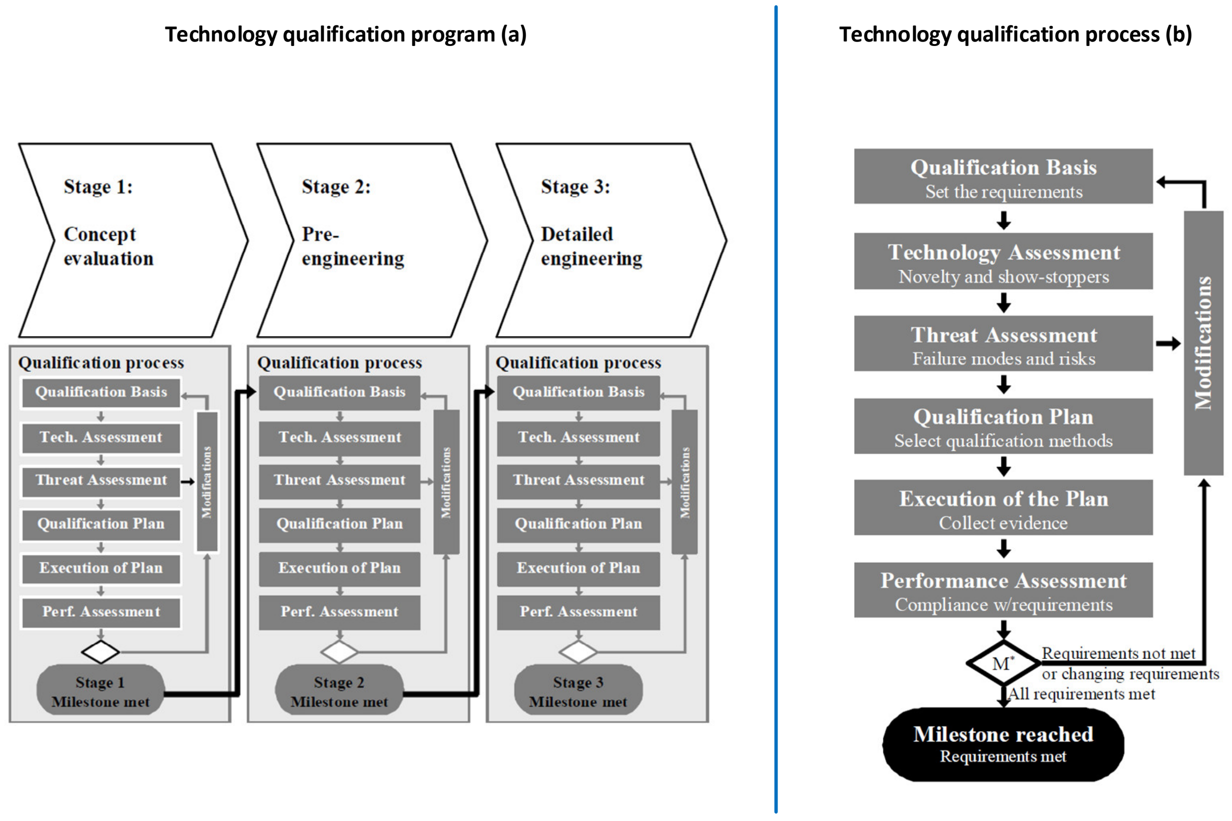


Figure 6-3. Technology qualification program (DNV-RP-A203, 2011).

The final step for each stage of the iteration is a performance assessment with the goal of measuring success by reviewing the qualification evidence vs the qualification basis (requirements). In the final iteration, during detail engineering, this implies confirmation that the technology meets all its requirements and that risk and uncertainty have been reduced to acceptable levels.

The results from the technology qualification can be used as:

- Acceptance for implementation of new technology.
- Basis for comparing different alternatives.
- Input for the evaluation of reliability of a larger system.
- Documentation of technology qualification development stage.
- In documenting regulatory compliance.

6.1.3 Dynamic reliability in technology qualification process

Hybrid models for dynamic reliability can be used in a technology qualification process to provide confidence of safe, reliable and optimal fault tolerant performance. In a short manner, the qualification process is a systematic guide in which:

- The requirements of the system are set.
- The technology is assessed to identify the novel elements and uncertainty around them.
- Failure modes and mechanisms are identified and their risk is assessed.
- The methods to provide evidence are specified.
- Evidence is collected.
- Performance assessments are carried out.

This is an iterative process as shown in Figure 6-3. Modifications are to be made if the requirements are not met and the milestone is only reached when all requirements are met.

Traditionally, qualification methods such as process simulation models, finite element methods are done separately than the reliability assessment using traditional methods such as reliability block diagrams and fault trees. Hybrid models are a mean to combine such models, for dynamic reliability/performance assessments that may allow to identify how failures or faults in components may lead to failures of the overall system, or to provide enough confidence that they do not. For example, it could be possible to identify if a fault tolerant control is able to keep the desired system performance under the presence of faults, with a quantitative target.

Qualification basis

The technology qualification basis aims to provide the common set of criteria against which the qualification activities and decisions will be assessed. In this step of the qualification process, the technology is completely described, through text, calculation data, drawings and other relevant documents. The functional requirements are specified quantitatively if possible. This includes reliability targets.

<u>Requirements of performance in fault tolerant control</u>

Different regions of requirements of performance can be identified for fault tolerant control as presented by Blanke et al. (2016) and shown in Figure 6-4.

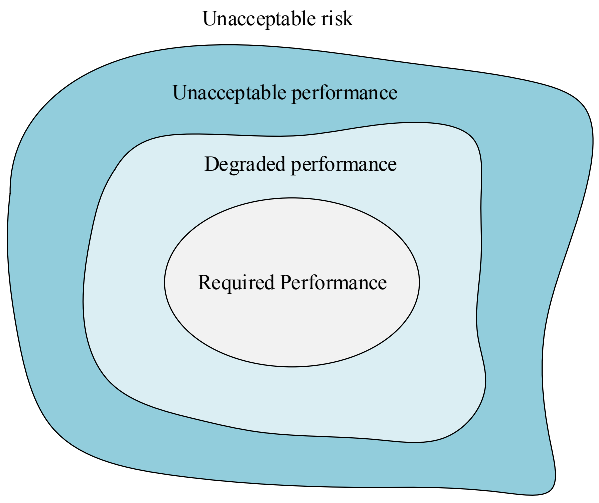


Figure 6-4. Regions of required, degraded and unacceptable performance. (Adopted from Blanke et al. (2016))

In the region of required performance, the system is able to satisfy its function. The system should ideally remain in this region. The controller can make the system remain in this region despite disturbances, uncertainties of the model and small faults like degrading components.

Faults can cause the system to go from the required performance region to the degraded performance. A fault tolerant controller must be able to take recovery actions to take the system back to the required performance region. A supervision level is then required at the border of these regions to diagnose the faults and adjust the controller.

The unacceptable performance region should be avoided by the means of a fault tolerant control. In this region, if the performance continues to deteriorate then a region of unacceptable risk may be reached. A safety system is enabled at this borderline to avoid hazardous events for the system and environment. In subsea processing facilities, safety systems and control systems are implemented as separate units, making it possible to design fault tolerant controllers without the need to meet safety standards.

Requirements of performance in gravity oil/gas separator

In the case study of oil/water gravity separators, these regions of performance can be associated with different level regions described by Arntzen (2016).

For process calculations, a level set point is assumed and called a normal level (NLL). This corresponds to the required performance region.

The separator is expected to produce outlet flows with certain specified qualities, depending on the downstream process. If the levels deviate too much form their intended set points, the performance may be at risk and the separator cannot produce acceptable outflows. At this point, the system reaches the unacceptable risk region. The surrounding must be protected due to safety or asset protection reasons. Asset protection is implemented in order to protect equipment such as compressors or pumps. A safety/asset protection system can act for these events, with shutdown that can be required for a very low level (LALL) or a very high level (LAHH).

Before the level reaches the unacceptable risk region, alarm levels are set to notify the operations that the control system is unable to manage the level and actions are required. These alarm levels, a level alarm low (LAL) and a level alarm high (LAH), corresponds to the boundary between the degraded performance and the unacceptable performance regions.

The qualification basis may require the system to operate within the acceptable performance region, i.e. between specified LAL and LAH, with a quantitative target, e.g. the separator must be able to operate with a water level h_{water} between 1.9 and 2.1 meters during 10 years, with a probability of 90%, for a given time.

Performance assessment

Performance assessment requires a quantitative reliability assessment to be carried out if a reliability target is stated in the qualification basis. This quantitative assessment is based on failure modes established in the threat assessment. Common methods for estimation of the system reliability are mentioned in DNV-RP-A203 (2011):

- Reliability block diagrams (RBD).
- Fault tree analysis (FTA).
- Monte Carlo simulations of RBDs, FTs or more complex systems, using a suitable software tool.

These traditional approaches such as RBD and FTA rely on failures of components and/or subsystems in series or parallel to estimate the reliability of the overall system. In fault tolerant control the reliability of the overall system can be improved because the system remains operational after the occurrence of faults even though the reliability of the components is not changed. Moreover, the effect of faulty (degraded) components on the reliability of the overall system is not clearly assessed with traditional methods. To show confidence on this fact in technology qualification programs, hybrid models can be used for performance and reliability assessments through Monte Carlo simulations.

Figure 6-5 shows the evolution of one simulation of the hybrid model for degradation in the valve. In the figure, the effect of shocks (occurring at times in which the accumulated damage in the valve body increases) on the water outflow F_{water} is observed. It can be noticed that the control system is able to compensate for the degradation process and position the valve accordingly, to maintain F_{water} at the needed value. This allows to maintain the set point of water level h_{water} (2 meters) and keep the separator in a relatively steady state as shown in Figure 6-6. It can be said that for this particular simulation the control is robust enough to tolerate the fault in a passive way.

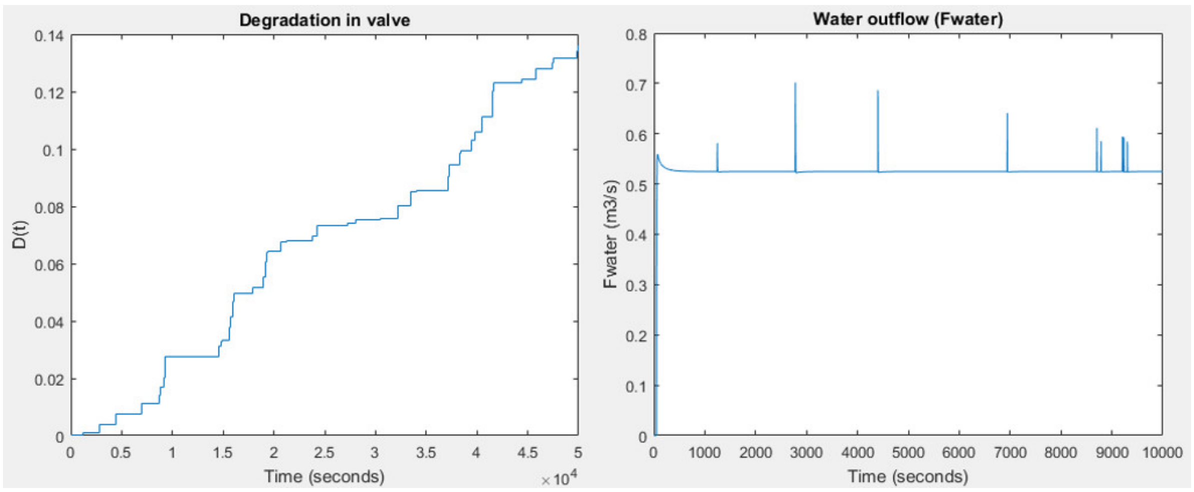


Figure 6-5. Degradation in valve and water outflow.

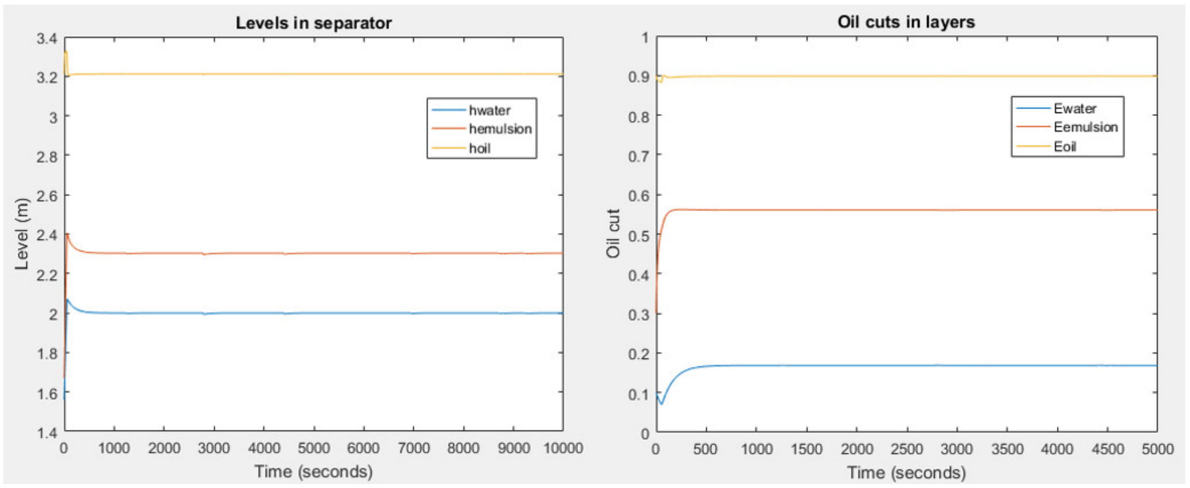


Figure 6-6. Levels in separator and oil cuts in layers.

Since the development of the fault/degradation follow a stochastic process, Monte Carlo simulations with several degradation paths are required for properly assessing the performance of the system during its mission time and support the decision process.

Figure 6-7 shows 50 simulations of the degradation in the valve and the water level under the presence of the degradation.

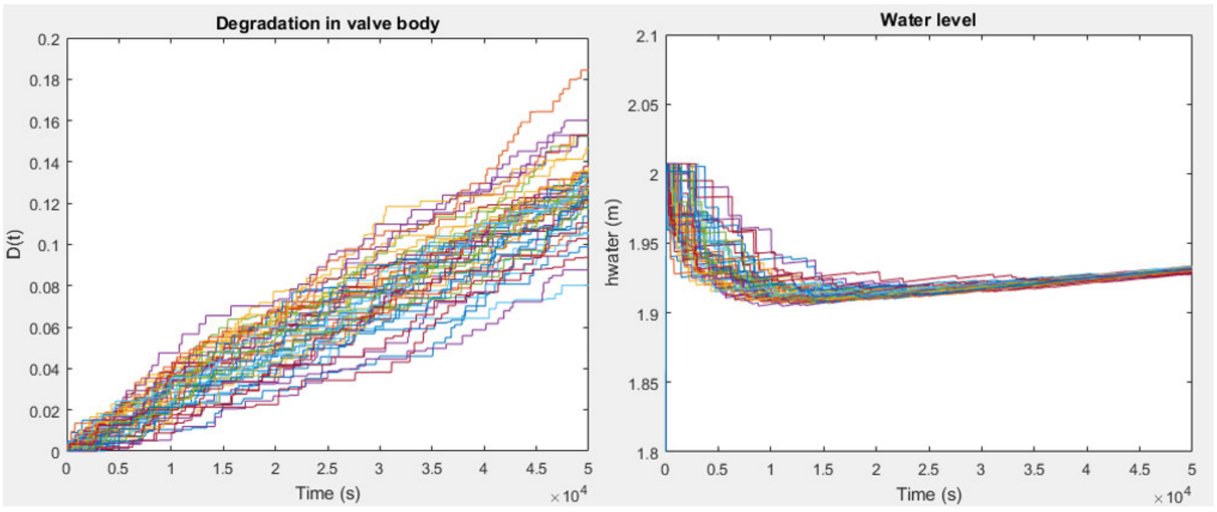


Figure 6-7. System performance under degrading valve.

In the figure, it can be observed that for the mission time, the system is able to perform within an acceptable region despite a degradation process occurring in the valve body. It is important to point, that several assumptions have been made in this model, such as constant inlet flow conditions to the separator, i.e. constant flow, oil cut and droplet size. Under an assumption like this, the model cannot be used to draw conclusions of a predicted performance, for that more uncertainties should be part of the model like disturbances in these conditions, which will require more advance control strategies, making it a challenging task to create appropriate hybrid dynamical models able to capture all dynamics involved in the system. However, the potential of hybrid modelling in the development of new technology for subsea applications and its qualification can be assimilated through the example presented based in the case study. The qualification process is a useful frame and its results may accept the implementation of the new technology, or serve the basis for comparing different alternatives, such as control and/or maintenance strategies for compact separators.

6.2 Application challenges

Despite the potential of hybrid models for coupling safety and reliability analysis with process and control in system design, for new technology development, there are many challenges to overcome. These challenges can be discussed in three different categories: modelling, simulation and analysis of results.

6.2.1 Modelling

Setting up a hybrid model creates the base over which the analysis is made, and it the first challenging task. Difficulties arise in understanding the dynamic operational conditions, the dynamic environment, and the dynamical interactions between components of a system, and describing everything by means of differential equations, that are later coupled with stochastic processes of failures and/or degradations.

Setting a hybrid model of a system, requires extensive knowledge about the system and often in different very technical fields like chemical engineering, control engineering, applied mathematics, computer science and others.

The gravity separator case study, represents a very simple case for a traditional reliability approaches like FT or RBD. Based on the physical decomposition presented in section 5.1, and Figure 5-1 a fault tree of can be modelled as in Figure 6-8.

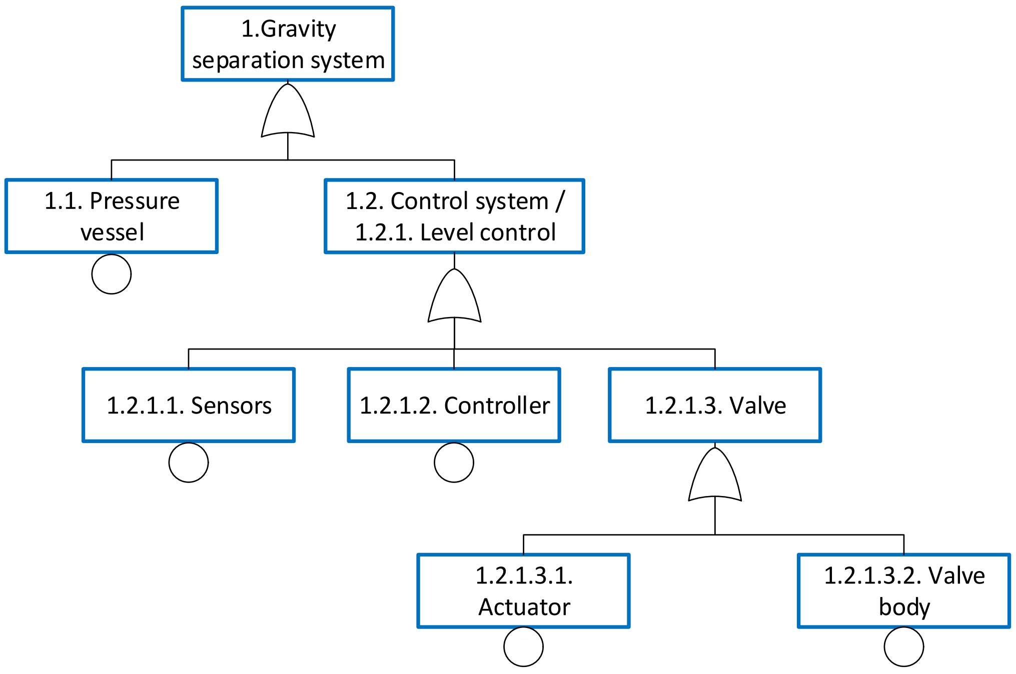


Figure 6-8. Fault tree of the gravity separation system.

The figure shows that from the traditional approach point of view, the system is very simple to model, since a failure of any component can cause the failure of the overall system. A hybrid model however, aims to capture much more than that and hence the system is much more complex to model, even with assumptions made in order to simplify it, making it more challenging for larger systems.

Hybrid models are specific application driven, in the sense that a model cannot be constructed in a general form, such as that they are easily adjusted for another application. In the gravity separator case study, the separation dynamics are dependent on many parameters, such as dimensions of the vessel, inlet flow rate, droplet size, oil cut, etc. and the control of this process depends on those parameters plus valve sizing and flow inherent characteristic, tuning parameters, etc.

6.2.2 Simulations

Since hybrid models are difficult to conceive and more to solve analytically, simulations are a mean to effectively solve the problem. However, the simulations required also find some difficulties.

One of the main challenges on the simulation of hybrid dynamic models occurs because of the different time scales in which the deterministic and stochastic processes evolve.

In the gravity separator case study, deterministic variables evolve in seconds, e.g. the inlet flow to the separator in the model is $F_{in} = 1.1\ {}^{m^3}\!/_{s}$ and the separator volume is of approximately $140\,m^3$ with ellipsoidal heads, meaning that the given inlet flow is capable of over filling a completely empty separator in a couple of minutes. Therefore, the balances between inlet flow and outflows require a control system that acts in seconds in order to maintain the level inside the separator. In this sense, numerical integration requires seconds as the time unit, and when using minutes the area calculations result in errors. The failures or degradation process of components do not occur that fast, in fact, high reliable components can last for years of operations without experiencing a failure, making seconds a not efficient unit for simulations of these processes.

As result, one simulation of the hybrid model can take very long times and generate large quantities of data not useful for analysis. Moreover, Monte Carlo simulations of the algorithm are required to estimate the reliability of the system. As presented in section 2.5, in their work, Manno et al. (2015) proposed two sampling engines in attempt to accelerate the discrete event simulation, and further simplifications are proposed based on a classification of the hybrid models according to their coupling and stochastic process type. However, developments in the simulations of hybrid models with large time scale differences are still needed to be able to perform dynamic reliability analysis effectively. Although the case study may be a bit unrealistic with large flow values, and how small the buffer of the separator is in relation to these, the large difference in times of occurrence of failures or degradation processes and the dynamics of a control system is clear.

6.2.3 Analysis and decisions

When results are obtained from a hybrid model involving many component functions, described in deterministic way, and many components failures or degradation processes, it is not so easy to make the appropriate decisions when improvements are required, in early stages of the technology development. While it may be simple to deduct whether or not the performance requirements are met, giving appropriate recommendations for improving the system is not so obvious.

Traditional approaches like RBD, allow analyzing the system structure and the structural importance of the components in a rather simple approach, making it simple to pinpoint the direction of improvement of the system and generate the appropriate recommendations in the performance assessment in a technology qualification process.

With a hybrid model is not as simple, and the analysis becomes a multidisciplinary task. Recommendations for improvement may go towards the control system or strategy, towards the process design, towards maintenance strategies or reliability of components. The analysis requires technical expertise of different fields, making it a task appropriate for a multidisciplinary team and systems integrator.

Based on the potential of hybrid models in dynamic safety and reliability analysis and its limitations, hybrid models and the dynamic reliability framework are not intended to replace traditional approaches, but to expand them, in ways that may allow the identification of problems not possible with traditional approaches, from early stages in the development of new technology.

Chapter 7

Guideline on hybrid modelling for dynamic reliability

This chapter presents a systematic approach to construct a hybrid model of a system, for dynamic reliability analysis. As discussed, hybrid models in dynamic reliability are constructed for specific applications, motivating the guideline described in this chapter. It provides useful tips that may facilitate the task of modelling and simulations, for a dynamic reliability problem. The approach builds over discussions and suggestions made in previous chapters, therefore references to earlier chapters are made under each step for more detailed information. Figure 7-1 shows the workflow of the proposed guideline in a logical flow.

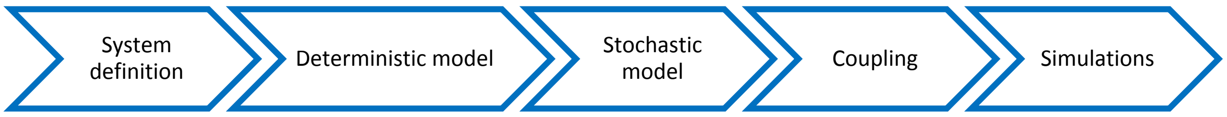


Figure 7-1. Dynamic reliability problems guideline.

7.1 System definition

As mentioned by Rauzy (2017), mastering the complexity of a system requires organizing and classifying its components, as well as characterizing their interactions. This is what a hybrid model aims to capture, integrating models from different engineering disciplines. The Krob architectural framework is an advanced methodology for model designs (Rauzy, 2017). The Krob architectural framework is a useful approach for hybrid modelling for dynamic reliability problems.

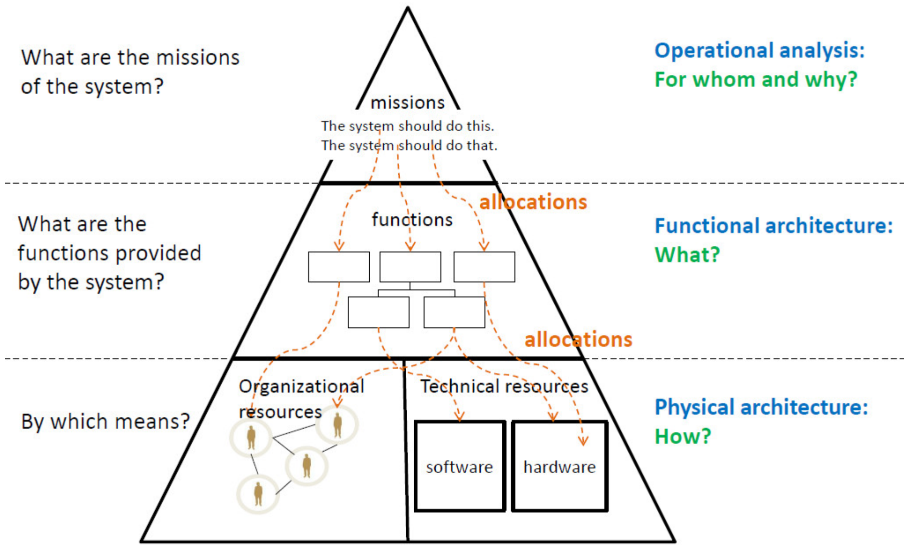


Figure 7-2. Krob architectural framework (Rauzy, 2017).

Figure 7-2 shows the architectural framework for modelling the system and describing it according to its operational analysis, its functional architecture and physical architecture. The characteristics and goals from these three points of view are summarized in Table 7-1, providing guidelines on how to describe the system with relevant keywords and questions.

Table 7-1. Krob architectural framework. (Adopted from Rauzy (2017))

Point of view	Questions	Analysis	Keywords	Models
Operational	For whom/what? Why?	Analysis of the environment of the system	Missions, use case, requirements, operational context, life cycle	Interactions of the system with its external environment
Functional	What?	Abstract analysis of the system	Function, task, process, mode	Abstract functions of the system
Physical	How?	Concrete analysis of the system	Component, part, architecture, configuration	Concrete components of the system

The operational analysis describes the overall systems missions, setting the requirements in performance and reliability that serve the basis of qualification on the development of new technology as described in section 6.1.3.

Based on the mission of the system to model, the system definition for hybrid modelling should describe the functional and physical architecture of the system, by means of decomposition.

Functional decomposition

Starting with a general mission of the system, this function is broken down into the closest sub-functions that need to work in order for the general function to work. These sub-functions may be further broken down and so on, until suitable. The result of the physical decomposition can be represented in a tree graph.

Physical decomposition

Similar to the functional decomposition, the system is decomposed into sub-systems that can be further decomposed into components, parts, until suitable for the model. This decomposition is based on physical systems, components and parts of the system. The result can also be represented as a tree graph.

Allocation

Sub-functions of the system are allocated to physical sub-systems, components or parts. One function may be allocated to more than one component that work together to achieve it, and in vice versa, more than one function can be achieved by one component.

The decomposition of the system in terms of functions and physical components and the allocation, provides a clear overview of the system to model. From the dynamic reliability perspective, stochastic failures and faults involve physical components of the system, affecting the ability of these to perform their allocated functions, described in terms of the physical laws as deterministic dynamic equations, and in vice versa these dynamic functions may affect the stochastic process of failures and faults of physical components.

7.2 Deterministic model

The deterministic model is focused on describing the sub-functions of the decomposition, in terms of the physical laws, i.e. rotational, thermodynamics, fluid mechanics, etc., by means of differential and/or algebraic equations.

Studying the physical laws is required to describe the function of components and interactions between them. The resulting equations usually include parameters from the characteristics of the components allocated to the functions, making the model specific for the application. The focus of study is how a component performs its allocated function and what physical law makes it possible and describes it.

It is suggested to study the physical laws and attempting to construct the deterministic model before the stochastic model, because it gives a better overview of the system. Describing the system by means of physical laws, may provide deeper knowledge on the functional and physical decomposition, requiring to iterate back to the system definition and delimitation.

7.3 Stochastic model

The stochastic model describes failures or degradation process of components. For this purpose, focus is placed on the physical decomposition and identifying failure mechanisms and/or modes of the components. These processes are described by stochastic models, as discrete-events or continuous.

Failures of components may be modelled as a Markov process while degradation can be modelled as continuous, using the appropriate distribution. Any other modelling formalisms for stochastic processes may also be used.

Up to this point, the deterministic model and stochastic models are done separately. They may even be constructed by different teams according to their field of expertise, making the system definition and decomposition very important for the integration that follows.

7.4 Coupling of models

Coupling the deterministic and stochastic model is an integration task. Based on the functional and physical decomposition, failures and or degradation process of components may influence the function or performance of these, and in vice versa, the dynamics to which the component is subjected may influence its failure rate or degradation process.

Shared variables of the deterministic and stochastic process are identified to couple the models. Usually, the influence of the stochastic process over the deterministic is achieved by a multiplicative or additive variable. For example, in the gravity separator case study, degradation in the valve body affects the flow across the valve as an addition ΔC_v to its flow coefficient, while the degradation of the actuator affects the ability to perform the control command in a multiplicative way. Failures may cause a full loss of a sub-function, for which a binary variable can be used as multiplicative to the equation or term describing the function, i.e. 1 for when the component works, 0 for when the component is failed.

At this stage, the type of coupling should be defined. In section 2.4.3, a classification is proposed as tight coupling or soft coupling. The choice may depend on the requirements of the system and the purpose of the model. Tight coupling is relevant for systems in which the dynamics have an influence over the stochastic process and enough information is available to relate the failure or degradation rates to the evolution of deterministic variables.

In discrete jumps, the transition rates may depend in the evolution of deterministic variables, to account for this, activation functions of the transitions should be defined. In degradation processes, one way to achieve a tight coupling is to use a proportional hazard model in which the degradation is a product of one factor dependent on time, and another factor depending on covariates that depend on the evolution of deterministic variables.

7.5 Simulations

Based on the coupling type and stochastic process, simulation algorithms are suggested in section 2.5. The choice of the simulation algorithm is illustrated in Figure 2-6. Simulink® is a suitable tool for the simulation of dynamic systems. MATLAB® can be used for Monte Carlo simulations of the Simulink® model, although it may not be the most efficient way. Other appropriate simulation tools should be considered and computer science expertise can be very useful.

Chapter 8

Summary and recommendations for further work

8.1 Summary and conclusions

Dynamic reliability is an extension of the traditional reliability theories, allowing to account for the dynamic behavior of most of the real-world industrial systems in safety and reliability assessments. Hybrid modelling presents a systematic approach to face a problem of dynamic reliability, in which the dynamic behavior of the system is described by means of a deterministic model that is later coupled with a stochastic model representing the failures or degradation process of components of the system. Hybrid models of this type find application from early stages of new technology development, integrating reliability analysis with systems design, to find optimal solutions with respect to safety, cost and performance while providing confidence in the qualification of such technology. Despite the potential of integrating reliability analysis with systems design, hybrid modelling and dynamic reliability problems do not come without challenges.

Constructing a hybrid model of a system requires technical knowledge in different fields of expertise, making it a multidisciplinary task that attempts to combine analyses that are traditionally done separately such as process design, control design, and safety and reliability analyses. Although the gravity separator case study presented is a rather simple system, many simplifications and assumptions are made, e.g. a single control loop, constant inlet flow conditions, densities of the liquid layers are assumed constant regardless of the oil cut, constant pressure drop across the valve regardless of its position or dynamic response to the control law, etc. These simplifications and assumptions of the model may limit the potential for achieving results that represent the real behavior of the system and model validation is a very complicated task, especially because stochastic processes are part of the dynamic model.

Subsea developments require high reliable solutions to make up for the difficulty to access the system for regular maintenance and quick repairs. These high reliable solutions mean that system failures are seldom and components are designed in a way that are highly resistant to deterioration or degradation processes, through appropriate materials, coatings, etc. On the other hand, research for subsea developments aims for more cost efficient solutions such as compact separators. Because of the reduced buffer in compact separators, they require advanced control strategies that are able to respond quickly so the mass and volume balances inside the separator are maintained. As result, the control and process dynamics evolve very fast, while the degradation process or failures evolve very slowly. This fact creates difficulties in the simulations of a hybrid model that combines these processes, requiring high computational power and more efficient algorithms, in order to make simulations of real cases an efficient task.

Simulink® is a convenient tool for modelling dynamic systems and its integration with MATLAB® allows to incorporate algorithms into the models and more than what is included in the block library, in a user friendly manner. Simulink® and MATLAB® are very popular academic tools and there is a lot of information available on the web, tutorials, discussion forums, and it is very easy to understand the basics. However, it may be possible that other tools are more efficient for simulations of hybrid models of the type of the separator case study of this work, overcoming some of the challenges of dynamic reliability problems.

The method chosen to approach the project, experiential learning, in which the focus was placed in a few relevant papers and case studies, to familiarize with the modelling procedure and tool, reflecting and then reinforcing with more literature review, was an efficient way to get a deeper understanding on hybrid modelling for dynamic reliability. This allowed to become familiar with the procedure and concept, before embarking into the task of constructing the separator case study, making it more efficient by knowing in what elements to focus for a hybrid model. By modelling different types of problems a better understanding of the capabilities of hybrid modelling for dynamic reliability assessments was gained, and the discussions and suggestions presented were not limited to the separator case study.

This project was conceived in a way that the subsea separation case study was suggested from the beginning, i.e. the original title on the project proposal was “Hybrid modelling for subsea separator”, without a very clear reason defining why to focus in a subsea separation process instead of other processing systems. In this sense, the project aimed to explore the potential of hybrid modelling for dynamic reliability assessments through a subsea separator case study, demonstrating relevant points, but the project was not conceived to solve a particular identified problem for which traditional approaches have proven to be not suitable. Because of this, in this project it is hard to demonstrate clearly with the case study, how dynamic reliability assessments could make the system more reliable and safer than with traditional approaches. Hybrid modelling is explored as a way to incorporate reliability and safety assessments with systems design from early stages of technology development, thus complementing but not replacing the traditional approaches that are still needed in the qualification process. It could be interesting to approach a project in the opposite direction, i.e. starting from an identified problem of a system, and exploring how hybrid modelling and dynamic reliability assessments could be applied to identify failure modes that traditional approaches have not been capable. It can be argued that traditional approaches are capable to do what dynamic reliability methods claim to do, by careful analysis and systems expertise, incorporating the dependency of components failures on system dynamics through the appropriate simplifications and performing analysis of different scenarios.

8.2 Recommendations for further work

This thesis focused on exploring the application of hybrid modelling and dynamic reliability assessment during systems design. It would be interesting to explore the application of hybrid modelling during subsea systems operations, and the potential of prognosis with remaining useful life estimations that account for the dynamic behavior of the system.

For systems operations, hybrid models have potentials that are worth studying such as:

- Through hybrid modelling, it is possible to integrate health monitoring with optimal control, in order to optimize the operation and maintenance strategies, according to the RUL estimations, to optimize production of subsea facilities.
- Through simulations of hybrid models, it is possible to gain a deeper understanding of how the degradation process of a component influences the performance of the system. This may allow the identification of new indicators for health monitoring purposes, that are based on the system performance in cases where more direct indicators of the health condition are not present.

In this thesis, hybrid modelling for dynamic reliability is presented from a general high level viewpoint and the corresponding discussions, potential and challenges are made based on the general approach. More particular cases of hybrid models such as Piecewise deterministic Markov process, switching diffusion process, and/or others, are worth studying in detail, to compare the capabilities and limitations of each, attempting to optimize the dynamic reliability assessments.

Bibliography

Alwi, H., Edwards, C., & Tan, C. P. (2011). *Fault Detection and Fault-Tolerant Control Using Sliding Modes*.

API-RP-553. (2012). Refinery Valves and Accessories for Control and Safety Instrumented Systems *API Recommended Practice 553*. Washington, DC: American Petroleum Institute Publishing Services.

Arntzen, R. (2001). *Gravity Separator Revamping.* (PhD Doctoral), Norwegian University of Science and Technology, Trondheim, Norway.

Arntzen, R. (2016). Savvy Separator: Level Design and Control in Gravity Separators. *Projects systems technologies. Oil and Gas Facilities, 5*.

Babykina, G., Brînzei, N., Aubry, J.-F., & Deleuze, G. (2016). Modeling and simulation of a controlled steam generator in the context of dynamic reliability using a Stochastic Hybrid Automaton. *Reliability Engineering & System Safety, 152*, 115-136. doi: http://dx.doi.org/10.1016/j.ress.2016.03.009

Blanke, M., Kinnaert, M., Lunze, J., & Staroswiecki, M. (2016). *Diagnosis and Fault-Tolerant Control* (3 ed.): Springer-Verlag Berlin Heidelberg.

Bujorianu, L. M. (2012). Hybrid Systems: Deterministic to Stochastic Perspectives *Stochastic Reachability Analysis of Hybrid Systems* (pp. pp 31-53): Springer London.

Campos, M. M. d., & Teixeira, H. (2010). *Controles típicos de equipamientos e processos industriais*. São Paulo, Brazil

Chiacchio, F., D'Urso, D., Compagno, L., Pennisi, M., Pappalardo, F., & Manno, G. (2016a). SHyFTA, a Stochastic Hybrid Fault Tree Automaton for the modelling and simulation of dynamic reliability problems. *Expert Systems with Applications, 47*, 42-57. doi: http://dx.doi.org/10.1016/j.eswa.2015.10.046

Chiacchio, F., D'Urso, D., Manno, G., & Compagno, L. (2016b). Stochastic hybrid automaton model of a multi-state system with aging: Reliability assessment and design consequences. *Reliability Engineering and System Safety, 149*, 1-13. doi: 10.1016/j.ress.2015.12.007

Cox, D. R. (1972). Regression Models and Life-Tables. *Journal of the Royal Statistical Society, 34*(2), 187-220.

Das, T., Backi, C., & Jäschke, J. (2017). A model for subsea oil-water gravity separator to estimate unmeasured disturbances (accepted / in print). In A. Espuña., M. Graells. & L. Puigjaner. (Eds.), *Computer Aided Chemical Engineering*: Elsevier.

DNV-GL. (2017). *Short-term Agility, Long-term Resilience. The outlook for the oil and gas industry in 2017.* https://www.dnvgl.com/oilgas/industry-outlook-report/short-term-agility-long-term-resilience.html

DNV-RP-A203. (2011). Qualification of New Technology: Det Norske Veritas AS.

Emerson. (2011). Control Valve Erosion Solutions. *Severe Service, 11*(4).

Emerson. (2014). Catalog 12. In F. C. International (Ed.).

FMC-Technologies. (2017). Subsea Processing Systems. Retrieved April 2017, from http://www.fmctechnologies.com/en/SubseaSystems/Technologies/SubseaProcessingSystems.aspx#

Henschke, M., Schlieper, L. H., & Pfennig, A. (2002). Determination of a coalescence parameter from batch-settling experiments. *Chemical Engineering Journal, 85*(2–3), 369-378. doi: http://doi.org/10.1016/S1385-8947(01)00251-0

Hu, J., Lygeros, J., & Sastry, S. (2000). Towards a Theory of Stochastic Hybrid Systems. In N. Lynch & B. H. Krogh (Eds.), *Hybrid Systems: Computation and Control: Third International Workshop, HSCC 2000 Pittsburgh, PA, USA, March 23–25, 2000 Proceedings* (pp. 160-173). Berlin, Heidelberg: Springer Berlin Heidelberg.

Kowalewski, S. (2002). Introduction to the Analysis and Verification of Hybrid Systems. In S. Engel, G. Frehse & E. Scnieder (Eds.), *Modelling, Analysis, and Design of Hybrid Systems*: Springer.

Labeau, P. E., Smidts, C., & Swaminathan, S. (2000). Dynamic reliability: towards an integrated platform for probabilistic risk assessment. *Reliability Engineering & System Safety, 68*(3), 219-254. doi: http://dx.doi.org/10.1016/S0951-8320(00)00017-X

Langeron, Y., Grall, A., & Barros, A. (2012). *Actuator lifetime management in industrial automation.* Paper presented at the IFAC Proceedings Volumes (IFAC-PapersOnline).

Langeron, Y., Grall, A., & Barros, A. (2015). A modeling framework for deteriorating control system and predictive maintenance of actuators. *Reliability Engineering and System Safety, 140*, 22-36. doi: 10.1016/j.ress.2015.03.028

Liu, Y., Zuo, M. J., Li, Y. F., & Huang, H. Z. (2015). Dynamic Reliability Assessment for Multi-State Systems Utilizing System-Level Inspection Data. *IEEE Transactions on Reliability, 64*(4), 1287-1299. doi: 10.1109/TR.2015.2418294

Lunze, J. (2002). What Is a Hybrid System? In S. Engel, G. Frehse & E. Scnieder (Eds.), *Modelling, Analysis, and Design of Hybrid Systems*: Springer.

Lygeros, J. (2007, July 16-19). *Stochastic hybrid systems* Paper presented at the 2nd HYCON PhD School on Hybrid Systems, Siena, Italy.

Manno, G., Zymaris, A., Kakalis, N. M. P., Chiacchio, F., Cipollone, F. E., Compagno, L., Urso, D. D., & Trapani, N. (2013). Dynamic reliability analysis of three nonlinear aging components with different failure modes characteristics *Safety, Reliability and Risk Analysis* (pp. 3047-3055): CRC Press.

Manno, G., Zymaris, A. S., Chiacchio, F., Compagno, L., & D'Urso, D. (2015). *Hybrid-pair modelling in dynamic reliability: Concepts, tool implementation and applications.* Paper presented at the Safety and Reliability of Complex Engineered Systems - Proceedings of the 25th European Safety and Reliability Conference, ESREL 2015.

MathWorks. (2016). Simulink. from https://se.mathworks.com/products/simulink/

Nguyen, D. N., Dieulle, L., & Grall, A. (2015). Remaining Useful Lifetime Prognosis of Controlled Systems: A Case of Stochastically Deteriorating Actuator. *Mathematical Problems in Engineering, 2015*, 16. doi: 10.1155/2015/356916

Nystad, B. H., Gola, G., Hulsund, J. E., & Roverso, D. (2010). *Technical condition assessment and remaining useful life estimation of choke valves subject to erosion.* Paper presented at the Annual Conference of the Prognostics and Health Management Society, PHM 2010.

Rausand, M., & Høyland, A. (2004). *System Reliability Theory. Models, Statistical Methods, and Applications* (Second ed.): John Wiley & Sons, Inc.

Rauzy, A. (2017). System architecture & design structure matrices. *NTNU Elements of Complex System Engineering Course Material / Lecture slides.*

Statoil. (2014). The subsea factory. Retrieved 22 september, 2016, from http://www.statoil.com/en/technologyinnovation/fielddevelopment/aboutsubsea/Pages/Lengre%20dypere%20kaldere.aspx

Statoil. (2016). Subsea Separator. *NTNU Subsea Production Systems Course Material / Lecture slides*.

SUBPRO. (2017). Report 2017. In J. Lippe (Ed.), *Subsea Production and Processing*. Norway: SFI Center for Research-based Innovation / The Research Council of Norway

whatispiping.com, W. (2016). Details about control valves. Retrieved May, 2017, from http://www.whatispiping.com/control-valves

Yang, H., Jiang, B., & Concquempot, V. (2010). *Fault Tolerant Control Design for Hybrid Systems* (1 ed.): Springer-Verlag Berlin Heidelberg.

Zhang, Y., Barros, A., & Rauzy, A. (2016). Assessment of a condition-based maintenance policy for Subsea systems: A preliminary study *Risk, Reliability and Safety: Innovating Theory and Practice* (pp. 1129-1136): CRC Press.

Appendix A - Acronyms

CAPEX: Capital expenditure.

FD: Fault diagnosis.

FT/FTA: Fault tree / Fault tree analysis.

FTC: Fault tolerant control.

HPP: Homogeneous Poisson Process.

HSE: Health, safety and environmental (management).

OPEX: Operational expenditure.

PDMP: Piecewise deterministic Markov process.

PDP: Piecewise deterministic process.

PHM: Proportional hazard model.

PID: Proportional, integral, derivative (controller). Note: controllers may only include some of the terms, e.g. P-controller (Proportional), PI-controller (Proportional, Integral).

RAMS: Reliability, availability, maintainability and safety.

RBD: Reliability block diagram.

RUL: Remaining useful lifetime.

SDP: Switching diffusion process.

Appendix B – C/Code generation

SIMULINK GENERATED C/CODE

```
/*
 * GravitySeparator.c
 * * Academic License - for use in teaching, academic research, and meeting
 * course requirements at degree granting institutions only.  Not for
 * government, commercial, or other organizational use.
 * * Code generation for model "GravitySeparator".
 * * Model version              : 1.84
 * Simulink Coder version : 8.11 (R2016b) 25-Aug-2016
 * C source code generated on : Sat Jun 10 18:23:55 2017
 * * Target selection: grt.tlc
 * Note: GRT includes extra infrastructure and instrumentation for prototyping
 * Embedded hardware selection: Intel->x86-64 (Windows64)
 * Code generation objectives: Unspecified
 * Validation result: Not run
 */
#include "GravitySeparator.h"
#include "GravitySeparator_private.h"
#define GravitySeparat_PoissonParameter (0.001)
/* Named constants for Chart: '<Root>/Effectiveness' */
#define GravitySepar_IN_NO_ACTIVE_CHILD ((uint8_T)0U)
#define GravitySeparator_IN_Phase_I ((uint8_T)1U)
#define GravitySeparator_IN_Phase_II ((uint8_T)2U)
/* Block signals (auto storage) */
B_GravitySeparator_T GravitySeparator_B;
/* Continuous states */
X_GravitySeparator_T GravitySeparator_X;
/* Block states (auto storage) */
DW_GravitySeparator_T GravitySeparator_DW;
/* Real-time model */
RT_MODEL_GravitySeparator_T GravitySeparator_M_;
RT_MODEL_GravitySeparator_T *const GravitySeparator_M = &GravitySeparator_M_;
/* Forward declaration for local functions */
static uint32_T GravitySeparato_eml_rand_str2id(void);
static void GravitySepar_eml_rand_mt19937ar(const uint32_T state[625], uint32_T
  state_0[625], real_T *r);
static real_T GravitySeparator_rand(DW_PoissonIntegerGenerator_Gr_T *localDW);
/*
 * This function updates continuous states using the ODE3 fixed-step
 * solver algorithm
 */
static void rt_ertODEUpdateContinuousStates(RTWSolverInfo *si )
{
  /* Solver Matrices */
  static const real_T rt_ODE3_A[3] = {
    1.0/2.0, 3.0/4.0, 1.0
  };
  static const real_T rt_ODE3_B[3][3] = {
    { 1.0/2.0, 0.0, 0.0 },

    { 0.0, 3.0/4.0, 0.0 },

    { 2.0/9.0, 1.0/3.0, 4.0/9.0 }
  };
  time_T t = rtsiGetT(si);
  time_T tnew = rtsiGetSolverStopTime(si);
  time_T h = rtsiGetStepSize(si);
  real_T *x = rtsiGetContStates(si);
  ODE3_IntgData *id = (ODE3_IntgData *)rtsiGetSolverData(si);
  real_T *y = id->y;
  real_T *f0 = id->f[0];
  real_T *f1 = id->f[1];
  real_T *f2 = id->f[2];
  real_T hB[3];
  int_T i;
  int_T nXc = 7;
  rtsiSetSimTimeStep(si,MINOR_TIME_STEP);
  /* Save the state values at time t in y, we'll use x as ynew. */
 (void) memcpy(y, x,
          (uint_T)nXc*sizeof(real_T));

 /* Assumes that rtsiSetT and ModelOutputs are up-
to-date */
 /* f0 = f(t,y) */
 rtsiSetdX(si, f0);
 GravitySeparator_derivatives();
 /* f(:,2) = feval(odefile, t + hA(1), y + f*hB(:,1),
args(:)(*)); */
 hB[0] = h * rt_ODE3_B[0][0];
 for (i = 0; i < nXc; i++) {
  x[i] = y[i] + (f0[i]*hB[0]);
 }
 rtsiSetT(si, t + h*rt_ODE3_A[0]);
 rtsiSetdX(si, f1);
 GravitySeparator_step();
 GravitySeparator_derivatives();
 /* f(:,3) = feval(odefile, t + hA(2), y + f*hB(:,2),
args(:)(*)); */
 for (i = 0; i <= 1; i++) {
  hB[i] = h * rt_ODE3_B[1][i];
 }
 for (i = 0; i < nXc; i++) {
  x[i] = y[i] + (f0[i]*hB[0] + f1[i]*hB[1]);
 }
 rtsiSetT(si, t + h*rt_ODE3_A[1]);
 rtsiSetdX(si, f2);
 GravitySeparator_step();
 GravitySeparator_derivatives();
 /* tnew = t + hA(3);
   ynew = y + f*hB(:,3); */
 for (i = 0; i <= 2; i++) {
  hB[i] = h * rt_ODE3_B[2][i];
 }
 for (i = 0; i < nXc; i++) {
  x[i] = y[i] + (f0[i]*hB[0] + f1[i]*hB[1] +
f2[i]*hB[2]);
 }

 rtsiSetT(si, tnew);
 rtsiSetSimTimeStep(si,MAJOR_TIME_STEP);
}
static                                  uint32_T
GravitySeparato_eml_rand_str2id(void)
{
 /* Start for MATLABSystem: '<S3>/Poisson
Integer Generator' */
 return 7U;
}
static                                      void
GravitySepar_eml_rand_mt19937ar(const uint32_T
state[625], uint32_T
 state_0[625], real_T *r)
{
 uint32_T u[2];
 uint32_T mti;
 uint32_T y;
 int32_T j;
 int32_T kk;

 /* Start for MATLABSystem: '<S3>/Poisson
Integer Generator' */
 memcpy(&state_0[0], &state[0], 625U *
sizeof(uint32_T));
 /*             ==========================
COPYRIGHT                           NOTICE
============================== */
 /* This is a uniform (0,1) pseudorandom number
generator based on:      */
 /* A C-program for MT19937, with initialization
improved 2002/1/26.     */
 /* Coded by Takuji Nishimura and Makoto
Matsumoto.              */
 /* Copyright (C) 1997 - 2002, Makoto Matsumoto
and Takuji Nishimura,     */
 /* All rights reserved.                       */
 /* Redistribution and use in source and binary
forms, with or without     */
 /* modification, are permitted provided that the
following conditions     */
 /* are met:                                   */
 /*   1. Redistributions of source code must retain
the above copyright    */
 /*   notice, this list of conditions and the following
disclaimer.    */
 /*     2. Redistributions in binary form must
reproduce the above copyright */
 /*   notice, this list of conditions and the following
disclaimer     */
 /*    in the documentation and/or other materials
provided with the     */
 /*    distribution.                              */
 /*   3. The names of its contributors may not be
used to endorse or      */
 /*    promote products derived from this software
without specific      */
 /*   prior written permission.                   */
 /* THIS SOFTWARE IS PROVIDED BY THE
COPYRIGHT HOLDERS AND CONTRIBUTORS
 /* "AS IS" AND ANY EXPRESS OR IMPLIED
WARRANTIES, INCLUDING, BUT NOT      */
 /* LIMITED TO, THE IMPLIED WARRANTIES
OF MERCHANTABILITY AND FITNESS FOR */
 /*    A PARTICULAR PURPOSE ARE
DISCLAIMED. IN NO EVENT SHALL THE
COPYRIGHT */
 /* OWNER OR CONTRIBUTORS BE LIABLE
FOR ANY DIRECT, INDIRECT, INCIDENTAL,
*/
 /*       SPECIAL, EXEMPLARY, OR
CONSEQUENTIAL DAMAGES (INCLUDING,
BUT NOT      */
 /*    LIMITED TO, PROCUREMENT OF
SUBSTITUTE GOODS OR SERVICES; LOSS OF
USE,  */
 /*   DATA, OR PROFITS; OR BUSINESS
INTERRUPTION) HOWEVER CAUSED AND
ON ANY
 /* THEORY OF LIABILITY, WHETHER IN
CONTRACT, STRICT LIABILITY, OR TORT   */
```

```
 /*       (INCLUDING       NEGLIGENCE    OR
OTHERWISE) ARISING IN ANY WAY OUT OF
THE USE */
 /*  OF THIS  SOFTWARE, EVEN IF ADVISED
OF THE POSSIBILITY OF SUCH DAMAGE.  */
 /* ==============================   END
================================== */
 do {
   for (j = 0; j < 2; j++) {
     mti = state_0[624] + 1U;
     if (mti >= 625U) {
       for (kk = 0; kk < 227; kk++) {
         y  =  (state_0[kk  +  1]  &  2147483647U)  |
(state_0[kk] & 2147483648U);
         if ((int32_T)(y & 1U) == 0) {
           y >>= 1U;
         } else {
           y = y >> 1U ^ 2567483615U;
         }
         state_0[kk] = state_0[kk + 397] ^ y;
       }
       for (kk = 0; kk < 396; kk++) {
         y  =  (state_0[kk  +  227]  &  2147483648U)  |
(state_0[kk + 228] &
           2147483647U);
         if ((int32_T)(y & 1U) == 0) {
           y >>= 1U;
         } else {
           y = y >> 1U ^ 2567483615U;
         }
         state_0[kk + 227] = state_0[kk] ^ y;
       }
       y    =    (state_0[623]    &    2147483648U)    |
(state_0[0] & 2147483647U);
       mti = 1U;
       if ((int32_T)(y & 1U) == 0) {
         y >>= 1U;
       } else {
         y = y >> 1U ^ 2567483615U;
       }
       state_0[623] = state_0[396] ^ y;
     }
     y = state_0[(int32_T)mti - 1];
     state_0[624] = mti;
     y ^= y >> 11U;
     y ^= y << 7U & 2636928640U;
     y ^= y << 15U & 4022730752U;
     y ^= y >> 18U;
     u[j] = y;
   }
   *r  =  ((real_T)(u[0]  >>  5U)  *  6.7108864E+7  +
(real_T)(u[1] >> 6U)) *
     1.1102230246251565E-16;
 } while (*r == 0.0);
}
real_T rt_roundd_snf(real_T u)
{
 real_T y;
 if (fabs(u) < 4.503599627370496E+15) {
   if (u >= 0.5) {
     y = floor(u + 0.5);
   } else if (u > -0.5) {
     y = u * 0.0;
   } else {
     y = ceil(u - 0.5);
   }
 } else {
   y = u;
 }
 return y;
}
static                                        real_T
GravitySeparator_rand(DW_PoissonIntegerGenerat
or_Gr_T *localDW)
{
 real_T r;
 uint32_T state[625];
 uint32_T INIT_FACTOR;
 uint32_T r_0;
 int32_T mti;
 /*  Start  for  MATLABSystem:  '<S3>/Poisson
Integer Generator' */
 if (localDW->method == 4U) {
   INIT_FACTOR = localDW->state / 127773U;
   r_0   =   (localDW->state   -   INIT_FACTOR   *
127773U) * 16807U;
   INIT_FACTOR *= 2836U;
   if (r_0 < INIT_FACTOR) {
     INIT_FACTOR  =  (r_0  -  INIT_FACTOR)  +
2147483647U;
   } else {
     INIT_FACTOR = r_0 - INIT_FACTOR;
   }
   localDW->state = INIT_FACTOR;
   r          =          (real_T)INIT_FACTOR          *
4.6566128752457969E-10;
 } else if (localDW->method == 5U) {
   INIT_FACTOR    =    69069U    *    localDW-
>state_n[0] + 1234567U;
   r_0  =  localDW->state_n[1]  <<  13  ^  localDW-
>state_n[1];
   r_0 ^= r_0 >> 17;
   r_0 ^= r_0 << 5;
   localDW->state_n[0] = INIT_FACTOR;
   localDW->state_n[1] = r_0;
   r    =    (real_T)(INIT_FACTOR    +    r_0)    *
2.328306436538696E-10;
 } else {
   if (!localDW->state_not_empty_c) {
     memset(&localDW->state_a[0],   0,   625U   *
sizeof(uint32_T));
     r_0 = 5489U;
     localDW->state_a[0] = 5489U;
     for (mti = 0; mti < 623; mti++) {
       r_0  =  (r_0  >>  30U  ^  r_0)  *  1812433253U  +
(int32_T)rt_roundd_snf((2.0 +
         (real_T)mti) - 1.0);
       localDW->state_a[mti + 1] = r_0;
     }
     localDW->state_a[624] = 624U;
```

```
    localDW->state_not_empty_c = true;
  }
  GravitySepar_eml_rand_mt19937ar(localDW-
>state_a, state, &r);
  memcpy(&localDW->state_a[0], &state[0],
625U * sizeof(uint32_T));
 }

 /* End of Start for MATLABSystem:
'<S3>/Poisson Integer Generator' */
 return r;
}
/*
 * Start for atomic system:
 *   'synthesized block'
 *   'synthesized block'
 */
void
G_PoissonIntegerGenerator_Start(DW_PoissonInte
gerGenerator_Gr_T *localDW)
{
 /* Start for MATLABSystem: '<S3>/Poisson
Integer Generator' */
 localDW->method =
GravitySeparato_eml_rand_str2id();
 localDW->method_not_empty = true;
 localDW->state = 1144108930U;
 localDW->state_not_empty = true;
 localDW->state_n[0] = 362436069U;
 localDW->state_n[1] = 521288629U;
 localDW->state_not_empty_f = true;
 localDW->obj.isInitialized = 0;
 localDW->objisempty = true;
 localDW->obj.isInitialized = 1;
}
/*
 * Output and update for atomic system:
 *   'synthesized block'
 *   'synthesized block'
 */
void
Gravity_PoissonIntegerGenerator(B_PoissonIntege
rGenerator_Gra_T *localB,
 DW_PoissonIntegerGenerator_Gr_T *localDW)
{
 real_T outTemp;
 real_T p;
 boolean_T whileFlag;
 real_T x;
 /* Start for MATLABSystem: '<S3>/Poisson
Integer Generator' incorporates:
  *  MATLABSystem: '<S3>/Poisson Integer
Generator'
  */
 outTemp = 0.0;
 p = 0.0;
 whileFlag = true;
 while (whileFlag) {
  x = GravitySeparator_rand(localDW);
  p -= log(x);
  if (p > GravitySeparat_PoissonParameter) {
   whileFlag = false;
  } else {
   outTemp++;
  }
 }

 /* MATLABSystem: '<S3>/Poisson Integer
Generator' incorporates:
  * Start for MATLABSystem: '<S3>/Poisson
Integer Generator'
  */
 localB->PoissonIntegerGenerator = outTemp;
}
/*
 * Termination for atomic system:
 *   'synthesized block'
 *   'synthesized block'
 */
void
Gr_PoissonIntegerGenerator_Term(DW_PoissonInt
egerGenerator_Gr_T *localDW)
{
 /* Start for MATLABSystem: '<S3>/Poisson
Integer Generator' incorporates:
  * Terminate for MATLABSystem: '<S3>/Poisson
Integer Generator'
  */
 if (localDW->obj.isInitialized == 1) {
  localDW->obj.isInitialized = 2;
 }
 /* End of Start for MATLABSystem:
'<S3>/Poisson Integer Generator' */
}
/*
 * Output and update for atomic system:
 *   '<S6>/Cross-section Area emulsion'
 *   '<S6>/Cross-section Area oil'
 *   '<S6>/Cross-section Area water'
 */
void Gravit_CrosssectionAreaemulsion(real_T
rtu_h, real_T rtu_r,
 B_CrosssectionAreaemulsion_Gr_T *localB)
{
 /* MATLAB Function 'Gravity Separator/Cross-
section Area emulsion': '<S9>:1' */
 /* Calculates vertical cross section area inside a */
 /*  cylindrical gravity separator at height h */
 /* '<S9>:1:4' */
 localB->Ac = (acos(1.0 - rtu_h / rtu_r) * 2.0 -
sin(acos(1.0 - rtu_h / rtu_r) *
   2.0)) * (rtu_r * rtu_r / 2.0);

 /* Cross section area [m^2] */
}
/*
* Output and update for atomic system:
 *   '<S6>/Derivative Cross-section Area'
 *   '<S6>/Derivative Cross-section Area1'
 *   '<S6>/Derivative Cross-section Area2'
 */
void        Grav_DerivativeCrosssectionArea(real_T
rtu_devh, real_T rtu_r, real_T rtu_h,
  real_T  rtu_L,  B_DerivativeCrosssectionArea__T
*localB)
{
  /*      MATLAB       Function       'Gravity
Separator/Derivative Cross-section Area': '<S16>:1'
*/
  /*  Calculates derivative of vertical cross section
area inside a  */
  /*  cylindrical gravity separator at height h  */
  /* '<S16>:1:4' */
  localB->devVc = sqrt(2.0 * rtu_r * rtu_h - rtu_h *
rtu_h) * (rtu_L * rtu_devh *
    2.0);

  /*  Cross section area [m^2] */
}
/*
 * Output and update for atomic system:
 *    '<S6>/Transfer Area emulsion'
 *    '<S6>/Transfer Area water'
 */
void       GravitySep_TransferAreaemulsion(real_T
rtu_h, real_T rtu_r, real_T rtu_L,
  B_TransferAreaemulsion_Gravit_T *localB)
{
  /* MATLAB Function 'Gravity Separator/Transfer
Area emulsion': '<S22>:1' */
  /*  Calculates horizontal transfer area at interface of
phases  */
  /*  Inside a cylindrical gravity separator at height h
*/
  /* '<S22>:1:5' */
  localB->At  =  sqrt(2.0  *  rtu_r  *  rtu_h  -  rtu_h  *
rtu_h) * (2.0 * rtu_L);

  /*  Cross section area [m^2] */
}
real_T      rt_urand_Upu32_Yd_f_pw_snf(uint32_T
*u)
{
  uint32_T lo;
  uint32_T hi;
  /*  Uniform  random  number  generator  (random
number between 0 and 1)
     #define IA      16807                magic multiplier
= 7^5
     #define IM      2147483647              modulus =
2^31-1
     #define IQ      127773                   IM div IA
     #define IR      2836                     IM modulo IA
     #define   S               4.656612875245797e-10
reciprocal of 2^31-1
     test = IA * (seed % IQ) - IR * (seed/IQ)
     seed = test < 0 ? (test + IM) : test
     return (seed*S)
   */
  lo = *u % 127773U * 16807U;
  hi = *u / 127773U * 2836U;
  if (lo < hi) {
    *u = 2147483647U - (hi - lo);
  } else {
    *u = lo - hi;
  }
  return (real_T)*u * 4.6566128752457969E-10;
}
real_T rt_powd_snf(real_T u0, real_T u1)
{
  real_T y;
  real_T tmp;
  real_T tmp_0;
  if (rtIsNaN(u0) || rtIsNaN(u1)) {
    y = (rtNaN);
  } else {
    tmp = fabs(u0);
    tmp_0 = fabs(u1);
    if (rtIsInf(u1)) {
      if (tmp == 1.0) {
        y = (rtNaN);
      } else if (tmp > 1.0) {
        if (u1 > 0.0) {
          y = (rtInf);
        } else {
          y = 0.0;
        }
      } else if (u1 > 0.0) {
        y = 0.0;
      } else {
        y = (rtInf);
      }
    } else if (tmp_0 == 0.0) {
      y = 1.0;
    } else if (tmp_0 == 1.0) {
      if (u1 > 0.0) {
        y = u0;
      } else {
        y = 1.0 / u0;
      }
    } else if (u1 == 2.0) {
      y = u0 * u0;
    } else if ((u1 == 0.5) && (u0 >= 0.0)) {
      y = sqrt(u0);
    } else if ((u0 < 0.0) && (u1 > floor(u1))) {
      y = (rtNaN);
    } else {
      y = pow(u0, u1);
    }
  }
  return y;
}
/* Model step function */
void GravitySeparator_step(void)
{
  /* local block i/o variables */
  real_T rtb_Integrator;
  real_T rtb_Integrator1;
  real_T rtb_Integrator2;
  real_T rtb_CumulativeSum;
  real_T rtb_CumulativeSum_f;
  real_T Re_inf;
  real_T K_HR;
  real_T lambda;
  real_T Xi;
  int16_T c;
  real_T rtb_Product14;
  real_T rtb_Fout;
  if (rtmIsMajorTimeStep(GravitySeparator_M)) {
    /* set solver stop time */
    if                    (!(GravitySeparator_M-
>Timing.clockTick0+1)) {
      rtsiSetSolverStopTime(&GravitySeparator_M-
>solverInfo,
                    ((GravitySeparator_M-
>Timing.clockTickH0 + 1) *
         GravitySeparator_M->Timing.stepSize0        *
4294967296.0));
    } else {
      rtsiSetSolverStopTime(&GravitySeparator_M-
>solverInfo,
                    ((GravitySeparator_M-
>Timing.clockTick0 + 1) *
         GravitySeparator_M->Timing.stepSize0 +
         GravitySeparator_M->Timing.clockTickH0 *
         GravitySeparator_M->Timing.stepSize0        *
4294967296.0));
    }
  }                          /* end MajorTimeStep */
  /* Update absolute time of base rate at minor time
step */
  if (rtmIsMinorTimeStep(GravitySeparator_M)) {
    GravitySeparator_M->Timing.t[0]             =
rtsiGetT(&GravitySeparator_M->solverInfo);
  }
  if (rtmIsMajorTimeStep(GravitySeparator_M)) {

Gravity_PoissonIntegerGenerator(&GravitySeparat
or_B.PoissonIntegerGenerator,

&GravitySeparator_DW.PoissonIntegerGenerator);
    /*    UniformRandomNumber:    '<S3>/Random
Number' */
    rtb_CumulativeSum_f                         =
GravitySeparator_DW.RandomNumber_NextOutp
ut;
    /* Product: '<S3>/Product' */
    rtb_CumulativeSum_f *=

GravitySeparator_B.PoissonIntegerGenerator.Poiss
onIntegerGenerator;
    /*          S-Function          (sdspcumsumprod):
'<S3>/Cumulative Sum' */
    c = 0;
    while (c < 1) {
      rtb_Fout                                  =
GravitySeparator_DW.CumulativeSum_RunningC
umVal;
      c = 0;
      while (c < 1) {
        rtb_Fout += rtb_CumulativeSum_f;
        rtb_CumulativeSum = rtb_Fout;
        c = 1;
      }

GravitySeparator_DW.CumulativeSum_RunningC
umVal = rtb_Fout;
      c = 1;
    }
    /*    End    of    S-Function    (sdspcumsumprod):
'<S3>/Cumulative Sum' */
    /* Chart: '<Root>/Effectiveness' */
    /* Gateway: Effectiveness */
    /* During: Effectiveness */
    if
(GravitySeparator_DW.is_active_c8_GravitySepar
ator == 0U) {
      /* Entry: Effectiveness */

GravitySeparator_DW.is_active_c8_GravitySeparat
or = 1U;
      /* Entry Internal: Effectiveness */
      /* Transition: '<S5>:7' */
      GravitySeparator_DW.is_c8_GravitySeparator
= GravitySeparator_IN_Phase_I;

      /* Entry 'Phase_I': '<S5>:1' */
      GravitySeparator_B.ef = 1.0;
    }                    else                    if
(GravitySeparator_DW.is_c8_GravitySeparator ==
              GravitySeparator_IN_Phase_I) {
      /* During 'Phase_I': '<S5>:1' */
      if (rtb_CumulativeSum > 0.032) {
        /* Transition: '<S5>:8' */
        GravitySeparator_DW.is_c8_GravitySeparator
=
          GravitySeparator_IN_Phase_II;
        /* Entry 'Phase_II': '<S5>:2' */
        GravitySeparator_B.ef       =       1.0       -
(rtb_CumulativeSum - 0.032) * 4.716;
      }
    } else {
      /* During 'Phase_II': '<S5>:2' */
      if (rtb_CumulativeSum > 0.032) {
        /* Transition: '<S5>:10' */
        GravitySeparator_DW.is_c8_GravitySeparator
=
          GravitySeparator_IN_Phase_II;
        /* Entry 'Phase_II': '<S5>:2' */
        GravitySeparator_B.ef       =       1.0       -
(rtb_CumulativeSum - 0.032) * 4.716;
      }
    }
    /* End of Chart: '<Root>/Effectiveness' */
    Gravity_PoissonIntegerGenerator

(&GravitySeparator_B.PoissonIntegerGenerator_p,

&GravitySeparator_DW.PoissonIntegerGenerator_
p);
```

```
  /* UniformRandomNumber: '<S4>/Random Number' */
  rtb_CumulativeSum_f =
GravitySeparator_DW.RandomNumber_NextOutput_d;
  /* Product: '<S4>/Product' */
  rtb_CumulativeSum_f *=

GravitySeparator_B.PoissonIntegerGenerator_p.PoissonIntegerGenerator;
  /* S-Function (sdspcumsumprod): '<S4>/Cumulative Sum' */
  c = 0;
  while (c < 1) {
   rtb_Fout =
GravitySeparator_DW.CumulativeSum_RunningCumVal_b;
   c = 0;
   while (c < 1) {
    rtb_Fout += rtb_CumulativeSum_f;
    rtb_CumulativeSum_f = rtb_Fout;
    c = 1;
   }

GravitySeparator_DW.CumulativeSum_RunningCumVal_b = rtb_Fout;
   c = 1;
  }
  /* End of S-Function (sdspcumsumprod): '<S4>/Cumulative Sum' */
  /* Sum: '<S7>/Add' incorporates:
   * Constant: '<S7>/Kv des (theoretical)'
   */
  GravitySeparator_B.Add =
rtb_CumulativeSum_f +
   GravitySeparator_P.Kvdestheoretical_Value;

  /* Constant: '<S6>/hwater' */
  GravitySeparator_B.hwater =
GravitySeparator_P.hwater_Value;
 }
 /* Integrator: '<S6>/Integrator' */
 if (GravitySeparator_DW.Integrator_IWORK != 0)
{
  GravitySeparator_X.Integrator_CSTATE =
GravitySeparator_B.hwater;
 }
 rtb_Integrator =
GravitySeparator_X.Integrator_CSTATE;
 /* End of Integrator: '<S6>/Integrator' */
 /* SignalConversion: '<S6>/ConcatBufferAtLevelsConcatenateIn1' */
 GravitySeparator_B.LevelsConcatenate[0] =
rtb_Integrator;
 if (rtmIsMajorTimeStep(GravitySeparator_M)) {
  /* Constant: '<S6>/hemulsion' */
  GravitySeparator_B.hemulsion =
GravitySeparator_P.hemulsion_Value;
 }
 /* Integrator: '<S6>/Integrator1' */
 if (GravitySeparator_DW.Integrator1_IWORK !=
0) {
  GravitySeparator_X.Integrator1_CSTATE =
GravitySeparator_B.hemulsion;
 }
 rtb_Integrator1 =
GravitySeparator_X.Integrator1_CSTATE;
 /* End of Integrator: '<S6>/Integrator1' */
 /* SignalConversion: '<S6>/ConcatBufferAtLevelsConcatenateIn2' */
 GravitySeparator_B.LevelsConcatenate[1] =
rtb_Integrator1;
 if (rtmIsMajorTimeStep(GravitySeparator_M)) {
  /* Constant: '<S6>/hoil' */
  GravitySeparator_B.hoil =
GravitySeparator_P.hoil_Value;
 }
 /* Integrator: '<S6>/Integrator2' */
 if (GravitySeparator_DW.Integrator2_IWORK !=
0) {
  GravitySeparator_X.Integrator2_CSTATE =
GravitySeparator_B.hoil;
 }
 rtb_Integrator2 =
GravitySeparator_X.Integrator2_CSTATE;
 /* End of Integrator: '<S6>/Integrator2' */
 /* SignalConversion: '<S6>/ConcatBufferAtLevelsConcatenateIn3' */
 GravitySeparator_B.LevelsConcatenate[2] =
rtb_Integrator2;
 /* Sum: '<Root>/Sum' incorporates:
  * Constant: '<Root>/Constant'
  */
 rtb_Product14 =
GravitySeparator_B.LevelsConcatenate[0] -
  GravitySeparator_P.Constant_Value;
 /* MinMax: '<S7>/Min' incorporates:
  * Constant: '<S7>/Max flow m3//s'
  * Gain: '<S8>/Proportional Gain'
  * Integrator: '<S8>/Integrator'
  * Product: '<S1>/Product'
  * Product: '<S7>/Product'
  * Sum: '<S8>/Sum'
  */
 GravitySeparator_B.Min =
fmin((GravitySeparator_P.PIDController_P *
  rtb_Product14 +
GravitySeparator_X.Integrator_CSTATE_a) *
  GravitySeparator_B.ef *
GravitySeparator_B.Add,
  GravitySeparator_P.Maxflowm3s_Value);
 if (rtmIsMajorTimeStep(GravitySeparator_M)) {
  /* Constant: '<S6>/Ewater' */
  GravitySeparator_B.Ewater =
GravitySeparator_P.Ewater_Value;
 }
 /* Integrator: '<S6>/Integrator3' */
 if (GravitySeparator_DW.Integrator3_IWORK !=
0) {
  GravitySeparator_X.Integrator3_CSTATE =
GravitySeparator_B.Ewater;
 }
 GravitySeparator_B.Integrator3 =
GravitySeparator_X.Integrator3_CSTATE;
 /* End of Integrator: '<S6>/Integrator3' */
 /* SignalConversion: '<S6>/ConcatBufferAtOil
cuts ConcatenateIn1' */
 GravitySeparator_B.OilcutsConcatenate[0] =
GravitySeparator_B.Integrator3;
 if (rtmIsMajorTimeStep(GravitySeparator_M)) {
  /* Constant: '<S6>/Eemulsion' */
  GravitySeparator_B.Eemulsion =
GravitySeparator_P.Eemulsion_Value;
 }
 /* Integrator: '<S6>/Integrator6' */
 if (GravitySeparator_DW.Integrator6_IWORK !=
0) {
  GravitySeparator_X.Integrator6_CSTATE =
GravitySeparator_B.Eemulsion;
 }
 /* SignalConversion: '<S6>/ConcatBufferAtOil
cuts ConcatenateIn2' incorporates:
  * Integrator: '<S6>/Integrator6'
  */
 GravitySeparator_B.OilcutsConcatenate[1] =
  GravitySeparator_X.Integrator6_CSTATE;
 if (rtmIsMajorTimeStep(GravitySeparator_M)) {
  /* Constant: '<S6>/Eoil' */
  GravitySeparator_B.Eoil =
GravitySeparator_P.Eoil_Value;
 }
 /* Integrator: '<S6>/Integrator7' */
 if (GravitySeparator_DW.Integrator7_IWORK !=
0) {
  GravitySeparator_X.Integrator7_CSTATE =
GravitySeparator_B.Eoil;
 }
 /* SignalConversion: '<S6>/ConcatBufferAtOil
cuts ConcatenateIn3' incorporates:
  * Integrator: '<S6>/Integrator7'
  */
 GravitySeparator_B.OilcutsConcatenate[2] =
  GravitySeparator_X.Integrator7_CSTATE;
 if (rtmIsMajorTimeStep(GravitySeparator_M)) {
 }
 /* MATLAB Function: '<S6>/MATLAB
Function1' incorporates:
  * Constant: '<S6>/Cd'
  * Constant: '<S6>/gravity'
  * Constant: '<S6>/hweir'
  * Constant: '<S6>/lweir'
  */
 /* MATLAB Function 'Gravity
Separator/MATLAB Function1': '<S20>:1' */
 /* '<S20>:1:2' */
 rtb_Fout = 0.66666666666666663 *
GravitySeparator_P.Cd_Value * sqrt(2.0 *
  GravitySeparator_P.gravity_Value) *
GravitySeparator_P.lweir_Value *
  rt_powd_snf(fmax(rtb_Integrator2 -
GravitySeparator_P.hweir_Value, 0.0), 1.5);
 /* MATLAB Function: '<S6>/Cross-section Area
oil' incorporates:
  * Constant: '<S6>/Radius'
  */

Gravit_CrosssectionAreaemulsion(rtb_Integrator2,
  GravitySeparator_P.Radius_Value,
&GravitySeparator_B.sf_CrosssectionAreaoil);
 /* MATLAB Function: '<S6>/Cross-section Area
emulsion' incorporates:
  * Constant: '<S6>/Radius'
  */

Gravit_CrosssectionAreaemulsion(rtb_Integrator1,
  GravitySeparator_P.Radius_Value,

&GravitySeparator_B.sf_CrosssectionAreaemulsio
n);
 if (rtmIsMajorTimeStep(GravitySeparator_M)) {
  /* MATLAB Function: '<S6>/Cross-section Area
weir' incorporates:
   * Constant: '<S6>/Radius'
   * Constant: '<S6>/hweir'
   */
  /* MATLAB Function 'Gravity Separator/Cross-
section Area weir': '<S12>:1' */
  /* Calculates vertical cross section area inside a
*/
  /* cylindrical gravity separator at height h */
  /* '<S12>:1:4' */
  GravitySeparator_B.Ac = (acos(1.0 -
GravitySeparator_P.hweir_Value /
   GravitySeparator_P.Radius_Value) * 2.0 -
sin(acos(1.0 -
   GravitySeparator_P.hweir_Value /
GravitySeparator_P.Radius_Value) * 2.0)) *
   (GravitySeparator_P.Radius_Value *
GravitySeparator_P.Radius_Value / 2.0);
  /* Cross section area [m^2] */
 }
 /* MATLAB Function: '<S6>/MATLAB
Function2' */
 /* MATLAB Function 'Gravity
Separator/MATLAB Function2': '<S21>:1' */
 /* '<S21>:1:3' */
 GravitySeparator_B.Foil = fmin(1.0, fmax(0.0,
  (GravitySeparator_B.sf_CrosssectionAreaoil.Ac -

GravitySeparator_B.sf_CrosssectionAreaemulsion.
Ac) /
  (GravitySeparator_B.sf_CrosssectionAreaoil.Ac -
GravitySeparator_B.Ac))) *
  rtb_Fout;
 if (rtmIsMajorTimeStep(GravitySeparator_M)) {
  /* ToWorkspace: '<Root>/To Workspace' */
  {
   double locTime = (((GravitySeparator_M-
>Timing.clockTick1+
                    GravitySeparator_M-
>Timing.clockTickH1* 4294967296.0))
                  );
     ;
     if  (rtmIsMajorTimeStep(GravitySeparator_M))
{
       rt_UpdateStructLogVar((StructLogVar *)

GravitySeparator_DW.ToWorkspace_PWORK.Log
gedData,
                       &locTime,
&GravitySeparator_B.Foil);
     }
   }
   /* ToWorkspace: '<Root>/To Workspace1' */
   {
     double    locTime    =    (((GravitySeparator_M-
>Timing.clockTick1+
                    GravitySeparator_M-
>Timing.clockTickH1* 4294967296.0))
                  );
     ;
     if  (rtmIsMajorTimeStep(GravitySeparator_M))
{
       rt_UpdateStructLogVar((StructLogVar *)

GravitySeparator_DW.ToWorkspace1_PWORK.Lo
ggedData,
                       &locTime,
&GravitySeparator_B.ef);
     }
   }
   /* ToWorkspace: '<Root>/To Workspace2' */
   {
     double    locTime    =    (((GravitySeparator_M-
>Timing.clockTick1+
                    GravitySeparator_M-
>Timing.clockTickH1* 4294967296.0))
                  );
     ;
     if  (rtmIsMajorTimeStep(GravitySeparator_M))
{
       rt_UpdateStructLogVar((StructLogVar *)

GravitySeparator_DW.ToWorkspace2_PWORK.Lo
ggedData,
                       &locTime,
&GravitySeparator_B.Min);
     }
   }
   /* ToWorkspace: '<Root>/To Workspace3' */
   if (rtmIsMajorTimeStep(GravitySeparator_M)) {
     rt_UpdateLogVar((LogVar *)(LogVar*)

(GravitySeparator_DW.ToWorkspace3_PWORK.L
oggedData),
                  &rtb_CumulativeSum, 0);
   }
   /* ToWorkspace: '<Root>/To Workspace4' */
   {
     double    locTime    =    (((GravitySeparator_M-
>Timing.clockTick1+
                    GravitySeparator_M-
>Timing.clockTickH1* 4294967296.0))
                  );
     ;
     if  (rtmIsMajorTimeStep(GravitySeparator_M))
{
       rt_UpdateStructLogVar((StructLogVar *)

GravitySeparator_DW.ToWorkspace4_PWORK.Lo
ggedData,
                       &locTime,
&GravitySeparator_B.OilcutsConcatenate[0]);
     }
   }
   /* ToWorkspace: '<Root>/To Workspace5' */
   {
     double    locTime    =    (((GravitySeparator_M-
>Timing.clockTick1+
                    GravitySeparator_M-
>Timing.clockTickH1* 4294967296.0))
                  );
     ;
     if  (rtmIsMajorTimeStep(GravitySeparator_M))
{
       rt_UpdateStructLogVar((StructLogVar *)

GravitySeparator_DW.ToWorkspace5_PWORK.Lo
ggedData,
                       &locTime,
&GravitySeparator_B.LevelsConcatenate[0]);
     }
   }
  /* ToWorkspace: '<Root>/To Workspace6' */
   if (rtmIsMajorTimeStep(GravitySeparator_M)) {
     rt_UpdateLogVar((LogVar *)(LogVar*)

(GravitySeparator_DW.ToWorkspace6_PWORK.L
oggedData),
                  &rtb_CumulativeSum_f, 0);
   }
 }
 /* Gain: '<S8>/Integral Gain' */
 GravitySeparator_B.IntegralGain              =
GravitySeparator_P.PIDController_I *
   rtb_Product14;
 if (rtmIsMajorTimeStep(GravitySeparator_M)) {
   /* ToWorkspace: '<S6>/To Workspace7' */
   if (rtmIsMajorTimeStep(GravitySeparator_M)) {
     rt_UpdateLogVar((LogVar *)(LogVar*)

(GravitySeparator_DW.ToWorkspace7_PWORK.L
oggedData),
                  &GravitySeparator_B.Integrator3, 0);
   }
   /*     MATLAB     Function:     '<S6>/MATLAB
Function' incorporates:
    *  Constant: '<S6>/Ein'
    *  Constant: '<S6>/Rho_oil'
   *  Constant: '<S6>/Rho_water'
   *  Constant: '<S6>/d'
   *  Constant: '<S6>/vis_oil'
   *  Constant: '<S6>/vis_water'
  /*          MATLAB          Function          'Gravity
Separator/MATLAB Function': '<S19>:1' */
  /* '<S19>:1:2' */
  /* '<S19>:1:3' */
  rtb_Product14                                     =
(GravitySeparator_P.Rho_water_Value -
               GravitySeparator_P.Rho_oil_Value) *
    GravitySeparator_P.Rho_oil_Value   *   9.81   *
rt_powd_snf
    (GravitySeparator_P.d_Value,          3.0)          /
(GravitySeparator_P.vis_oil_Value *
    GravitySeparator_P.vis_oil_Value);
  /*  Archimedes number */
  /* '<S19>:1:4' */
  Re_inf  =  (rt_powd_snf(0.01  *  rtb_Product14  +
1.0, 0.5714285714285714) - 1.0)
    * 9.72;
  /*  Reynolds number */
  /* '<S19>:1:5' */
  K_HR = (GravitySeparator_P.vis_oil_Value +
        GravitySeparator_P.vis_water_Value) * 3.0 /
(2.0 *
    GravitySeparator_P.vis_oil_Value + 3.0 *
    GravitySeparator_P.vis_water_Value);
  /*  Hadamard Rybczynski factor    */
  /* '<S19>:1:6' */
  lambda = (1.0 - GravitySeparator_P.Ein_Value) /
2.0 /
    GravitySeparator_P.Ein_Value    /    K_HR     *
exp(2.5 *
    GravitySeparator_P.Ein_Value  /  (1.0  -  0.61   *
GravitySeparator_P.Ein_Value));
  /* '<S19>:1:7' */
  Xi = rt_powd_snf(GravitySeparator_P.Ein_Value
/ (1.0 -
    GravitySeparator_P.Ein_Value),  0.45)  *  (5.0  /
rt_powd_snf(K_HR, 1.5));

  /* '<S19>:1:8' */
  Re_inf = rtb_Product14 / 6.0 / (Re_inf * Re_inf) -
3.0 / K_HR / Re_inf;
  /* '<S19>:1:9' */
  /* '<S19>:1:10' */
  GravitySeparator_B.v_u  =  (sqrt(rtb_Product14  *
Re_inf * Xi * rt_powd_snf(1.0
    -  GravitySeparator_P.Ein_Value,  3.0)  /  54.0  /
(lambda * lambda) /
    (GravitySeparator_P.Ein_Value                    *
GravitySeparator_P.Ein_Value) + 1.0) - 1.0)
    *          (3.0          *          lambda          *
GravitySeparator_P.Ein_Value / Re_inf / Xi / (1.0 -
      GravitySeparator_P.Ein_Value))                  *
GravitySeparator_P.vis_oil_Value /
    GravitySeparator_P.Rho_oil_Value                 /
GravitySeparator_P.d_Value;
 }
```

```
  /*   MATLAB   Function:   '<S6>/Transfer   Area
emulsion' incorporates:
   *  Constant: '<S6>/Length'
   *  Constant: '<S6>/Radius'
   */

GravitySep_TransferAreaemulsion(rtb_Integrator1,
   GravitySeparator_P.Radius_Value,
GravitySeparator_P.Length_Value,

&GravitySeparator_B.sf_TransferAreaemulsion);
  /* Product: '<S6>/Product14' incorporates:
   *  Constant: '<S6>/Pfup'
   */
  rtb_Product14 = GravitySeparator_P.Pfup_Value *
GravitySeparator_B.v_u *

GravitySeparator_B.sf_TransferAreaemulsion.At;
  /* Product: '<S6>/Product18' incorporates:
   *  Constant: '<S6>/Constant1'
   *  Constant: '<S6>/Pqup'
   *  Constant: '<S6>/hweir'
   *  MATLAB Function: '<S6>/k1 calc'
   *  MATLAB Function: '<S6>/vel_oil'
   *  MinMax: '<S6>/Max1'
   *  Product: '<S6>/Divide9'
   *  Product: '<S6>/Product17'
   *  Sum: '<S6>/Sum7'
   */
  /* MATLAB Function 'Gravity Separator/k1 calc':
'<S24>:1' */
  /* '<S24>:1:2' */
  /* MATLAB Function 'Gravity Separator/vel_oil':
'<S25>:1' */
  /* '<S25>:1:3' */
  lambda     =     fmax(fmin(sqrt(rtb_Integrator1      /
GravitySeparator_P.hweir_Value), exp
          ((rtb_Integrator1                              /
GravitySeparator_P.hweir_Value - 1.0) *
          -100.0))                                       *
GravitySeparator_B.sf_TransferAreaemulsion.At *
        ((rtb_Integrator1     -     rtb_Integrator)     /
rtb_Integrator2) *
        (GravitySeparator_B.Foil /

(GravitySeparator_B.sf_CrosssectionAreaoil.Ac    -
fmax

(GravitySeparator_B.sf_CrosssectionAreaemulsion.
Ac,
          GravitySeparator_B.Ac))),
GravitySeparator_P.Constant1_Value)
   * GravitySeparator_P.Pqup_Value;
  /* Sum: '<S6>/Sum8' */
  rtb_Fout -= GravitySeparator_B.Foil;
  if (rtmIsMajorTimeStep(GravitySeparator_M)) {
   /* Product: '<S14>/Product' incorporates:
    *  Constant: '<S14>/Constant1'
    *  Constant: '<S6>/Length'
    */
  GravitySeparator_B.Product                        =
GravitySeparator_P.Constant1_Value_e *
    GravitySeparator_P.Length_Value;
 }
 /* Sum: '<S14>/Sum' incorporates:
  *  Constant: '<S14>/Constant'
  *  Constant: '<S6>/Radius'
  *  Math: '<S14>/Math Function'
  *  Product: '<S14>/Product1'
  */
 GravitySeparator_B.Sum                             =
GravitySeparator_P.Constant_Value_b *
  GravitySeparator_P.Radius_Value                   *
rtb_Integrator1 - rtb_Integrator1 *
  rtb_Integrator1;
 /* Product: '<S6>/Divide1' incorporates:
  *  Constant: '<S6>/Fin'
  *  Product: '<S14>/Product2'
  *  Sqrt: '<S14>/Sqrt'
  *  Sum: '<S6>/Sum3'
  */
 GravitySeparator_B.Divide1                         =
((((GravitySeparator_P.Fin_Value - rtb_Product14)
  - lambda) - GravitySeparator_B.Min) - rtb_Fout)
/
  (GravitySeparator_B.Product                       *
sqrt(GravitySeparator_B.Sum));
 /* MATLAB Function: '<S6>/Derivative Cross-
section Area1' incorporates:
  *  Constant: '<S6>/Length'
  *  Constant: '<S6>/Radius'
  */

Grav_DerivativeCrosssectionArea(GravitySeparato
r_B.Divide1,
  GravitySeparator_P.Radius_Value,
rtb_Integrator,
  GravitySeparator_P.Length_Value,

&GravitySeparator_B.sf_DerivativeCrosssectionAr
ea1);
 if (rtmIsMajorTimeStep(GravitySeparator_M)) {
  /* Product: '<S6>/Divide6' incorporates:
   *  Constant: '<S6>/d'
   *  Constant: '<S6>/tcoalescence'
   */
  GravitySeparator_B.Divide6                        =
GravitySeparator_P.d_Value /
    GravitySeparator_P.tcoalescence_Value;
 }
 /* MATLAB Function: '<S6>/Transfer Area water'
incorporates:
  *  Constant: '<S6>/Length'
  *  Constant: '<S6>/Radius'
  */
 GravitySep_TransferAreaemulsion(rtb_Integrator,
  GravitySeparator_P.Radius_Value,
GravitySeparator_P.Length_Value,
  &GravitySeparator_B.sf_TransferAreawater);
 /* MATLAB Function: '<S6>/Cross-section Area
water' incorporates:
  *  Constant: '<S6>/Radius'
  */
 Gravit_CrosssectionAreaemulsion(rtb_Integrator,
  GravitySeparator_P.Radius_Value,

&GravitySeparator_B.sf_CrosssectionAreawater);
 /* Product: '<S6>/Product16' incorporates:
  *  Constant: '<S6>/Constant'
  *  Constant: '<S6>/Pqdown'
  *  MinMax: '<S6>/Max'
  *  Product: '<S6>/Divide7'
  *  Product: '<S6>/Divide8'
  *  Product: '<S6>/Product15'
  *  Sum: '<S6>/Sum'
  */
 Xi = fmax(GravitySeparator_B.Min /

GravitySeparator_B.sf_CrosssectionAreawater.Ac *

GravitySeparator_B.sf_TransferAreawater.At          *
((rtb_Integrator1 -
       rtb_Integrator) / rtb_Integrator2),
      GravitySeparator_P.Constant_Value_k) *
  GravitySeparator_P.Pqdown_Value;
 if (rtmIsMajorTimeStep(GravitySeparator_M)) {
  /* Product: '<S13>/Product' incorporates:
   *  Constant: '<S13>/Constant1'
   *  Constant: '<S6>/Length'
   */
  GravitySeparator_B.Product_j                      =
GravitySeparator_P.Constant1_Value_eq *
    GravitySeparator_P.Length_Value;
 }
 /* Sum: '<S13>/Sum' incorporates:
  *  Constant: '<S13>/Constant'
  *  Constant: '<S6>/Radius'
  *  Math: '<S13>/Math Function'
  *  Product: '<S13>/Product1'
  */
 GravitySeparator_B.Sum_e                           =
GravitySeparator_P.Constant_Value_a *
  GravitySeparator_P.Radius_Value                   *
rtb_Integrator - rtb_Integrator *
  rtb_Integrator;
 /* Product: '<S6>/Divide' incorporates:
  *  Constant: '<S6>/Constant2'
  *  Constant: '<S6>/Pfdown'
  *  Product: '<S13>/Product2'
  *  Product: '<S6>/Product13'
  *  Sqrt: '<S13>/Sqrt'
  *  Sum: '<S6>/Sum1'
  */
 GravitySeparator_B.Divide                          =
((GravitySeparator_P.Pfdown_Value *
  GravitySeparator_P.Constant2_Value                *
GravitySeparator_B.Divide6 *
  GravitySeparator_B.sf_TransferAreawater.At        +
Xi) - GravitySeparator_B.Min) /
```

```
  (GravitySeparator_B.Product_j                    *
sqrt(GravitySeparator_B.Sum_e));
 /* MATLAB Function: '<S6>/Derivative Cross-
section Area' incorporates:
  * Constant: '<S6>/Length'
  * Constant: '<S6>/Radius'
  */

Grav_DerivativeCrosssectionArea(GravitySeparato
r_B.Divide,
  GravitySeparator_P.Radius_Value,
rtb_Integrator,
  GravitySeparator_P.Length_Value,

&GravitySeparator_B.sf_DerivativeCrosssectionAr
ea);
 /* Sum: '<S6>/Add' */
 Re_inf                                           =
GravitySeparator_B.sf_DerivativeCrosssectionArea
1.devVc -

GravitySeparator_B.sf_DerivativeCrosssectionArea
.devVc;
 if (rtmIsMajorTimeStep(GravitySeparator_M)) {
  /* Product: '<S15>/Product' incorporates:
   * Constant: '<S15>/Constant1'
   * Constant: '<S6>/Length'
   */
  GravitySeparator_B.Product_jw                   =
GravitySeparator_P.Constant1_Value_o *
   GravitySeparator_P.Length_Value;
 }
 /* Sum: '<S15>/Sum' incorporates:
  * Constant: '<S15>/Constant'
  * Constant: '<S6>/Radius'
  * Math: '<S15>/Math Function'
  * Product: '<S15>/Product1'
  */
 GravitySeparator_B.Sum_b                         =
GravitySeparator_P.Constant_Value_bh *
  GravitySeparator_P.Radius_Value                 *
rtb_Integrator2 - rtb_Integrator2 *
  rtb_Integrator2;
 /* Product: '<S6>/Divide2' incorporates:
  * Constant: '<S6>/Fin'
  * Product: '<S15>/Product2'
  * Sqrt: '<S15>/Sqrt'
  * Sum: '<S6>/Sum5'
  */
 GravitySeparator_B.Divide2                       =
(((GravitySeparator_P.Fin_Value -
  GravitySeparator_B.Foil)    -    rtb_Fout)    -
GravitySeparator_B.Min) /
  (GravitySeparator_B.Product_jw                  *
sqrt(GravitySeparator_B.Sum_b));
 /* MATLAB Function: '<S6>/Derivative Cross-
section Area2' incorporates:
  * Constant: '<S6>/Length'
  * Constant: '<S6>/Radius'
  */

Grav_DerivativeCrosssectionArea(GravitySeparato
r_B.Divide2,
  GravitySeparator_P.Radius_Value,
rtb_Integrator,
  GravitySeparator_P.Length_Value,

&GravitySeparator_B.sf_DerivativeCrosssectionAr
ea2);
 if (rtmIsMajorTimeStep(GravitySeparator_M)) {
  /* Product: '<S6>/Product4' incorporates:
   * Constant: '<S6>/Ein'
   * Constant: '<S6>/Fin'
   */
  GravitySeparator_B.Product4                     =
GravitySeparator_P.Ein_Value *
   GravitySeparator_P.Fin_Value;
 }
 /* Product: '<S6>/Divide3' incorporates:
  * Constant: '<S6>/Length'
  * Integrator: '<S6>/Integrator6'
  * Product: '<S6>/Product'
  * Product: '<S6>/Product1'
  * Product: '<S6>/Product2'
  * Product: '<S6>/Product3'
  * Sum: '<S6>/Sum2'
  */
 GravitySeparator_B.Divide3                       =
((GravitySeparator_X.Integrator6_CSTATE * Xi -
  GravitySeparator_B.Integrator3                  *
GravitySeparator_B.Min) -
  GravitySeparator_B.Integrator3 *

GravitySeparator_B.sf_DerivativeCrosssectionArea
.devVc) /

(GravitySeparator_B.sf_CrosssectionAreawater.Ac
*
   GravitySeparator_P.Length_Value);
 /* Product: '<S6>/Divide4' incorporates:
  * Constant: '<S6>/Length'
  * Integrator: '<S6>/Integrator6'
  * Product: '<S6>/Product5'
  * Product: '<S6>/Product6'
  * Product: '<S6>/Product7'
  * Product: '<S6>/Product8'
  * Sum: '<S6>/Add1'
  * Sum: '<S6>/Add2'
  * Sum: '<S6>/Sum4'
  */
 GravitySeparator_B.Divide4                       =
((((GravitySeparator_B.Product4 - rtb_Product14)
  -   GravitySeparator_X.Integrator6_CSTATE    *
rtb_Fout) -
  GravitySeparator_X.Integrator6_CSTATE           *
Re_inf) - (lambda + Xi) *
  GravitySeparator_X.Integrator6_CSTATE) /

((GravitySeparator_B.sf_CrosssectionAreaemulsio
n.Ac -
GravitySeparator_B.sf_CrosssectionAreawater.Ac)
*
    GravitySeparator_P.Length_Value);
  /* Product: '<S6>/Divide5' incorporates:
   *  Constant: '<S6>/Length'
   *  Integrator: '<S6>/Integrator6'
   *  Integrator: '<S6>/Integrator7'
   *  Product: '<S6>/Product10'
   *  Product: '<S6>/Product11'
   *  Product: '<S6>/Product12'
   *  Product: '<S6>/Product9'
   *  Sum: '<S6>/Add3'
   *  Sum: '<S6>/Add4'
   *  Sum: '<S6>/Sum6'
   */
  GravitySeparator_B.Divide5                    =
(((GravitySeparator_X.Integrator6_CSTATE        *
lambda
   +                    rtb_Product14)              -
GravitySeparator_X.Integrator7_CSTATE *
   GravitySeparator_B.Foil) -

(GravitySeparator_B.sf_DerivativeCrosssectionAre
a2.devVc -

GravitySeparator_B.sf_DerivativeCrosssectionArea
1.devVc) *
   GravitySeparator_X.Integrator7_CSTATE) /
   ((GravitySeparator_B.sf_CrosssectionAreaoil.Ac
-

GravitySeparator_B.sf_CrosssectionAreaemulsion.
Ac) *
    GravitySeparator_P.Length_Value);
  if (rtmIsMajorTimeStep(GravitySeparator_M)) {
   /* Matfile logging */
   rt_UpdateTXYLogVars(GravitySeparator_M-
>rtwLogInfo,
                    (GravitySeparator_M->Timing.t));
  }                             /* end MajorTimeStep */
  if (rtmIsMajorTimeStep(GravitySeparator_M)) {
   if (rtmIsMajorTimeStep(GravitySeparator_M)) {
    /*    Update    for    UniformRandomNumber:
'<S3>/Random Number' */

GravitySeparator_DW.RandomNumber_NextOutp
ut =

(GravitySeparator_P.RandomNumber_Maximum -

GravitySeparator_P.RandomNumber_Minimum)   *
rt_urand_Upu32_Yd_f_pw_snf
      (&GravitySeparator_DW.RandSeed) +

GravitySeparator_P.RandomNumber_Minimum;
    /*    Update    for    UniformRandomNumber:
'<S4>/Random Number' */

GravitySeparator_DW.RandomNumber_NextOutp
ut_d =

(GravitySeparator_P.RandomNumber_Maximum_
m -

GravitySeparator_P.RandomNumber_Minimum_c)
* rt_urand_Upu32_Yd_f_pw_snf
       (&GravitySeparator_DW.RandSeed_b) +

GravitySeparator_P.RandomNumber_Minimum_c;
   }
   /* Update for Integrator: '<S6>/Integrator' */
   GravitySeparator_DW.Integrator_IWORK = 0;
   /* Update for Integrator: '<S6>/Integrator1' */
   GravitySeparator_DW.Integrator1_IWORK = 0;
   /* Update for Integrator: '<S6>/Integrator2' */
   GravitySeparator_DW.Integrator2_IWORK = 0;
   /* Update for Integrator: '<S6>/Integrator3' */
   GravitySeparator_DW.Integrator3_IWORK = 0;
   /* Update for Integrator: '<S6>/Integrator6' */
   GravitySeparator_DW.Integrator6_IWORK = 0;
   /* Update for Integrator: '<S6>/Integrator7' */
   GravitySeparator_DW.Integrator7_IWORK = 0;
  }                             /* end MajorTimeStep */
  if (rtmIsMajorTimeStep(GravitySeparator_M)) {
   /* signal main to stop simulation */
   {                            /* Sample time: [0.0s, 0.0s]
*/
    if     ((rtmGetTFinal(GravitySeparator_M)!=-1)
&&
      !((rtmGetTFinal(GravitySeparator_M)-
       (((GravitySeparator_M-
>Timing.clockTick1+
         GravitySeparator_M-
>Timing.clockTickH1* 4294967296.0)) )) >
       (((GravitySeparator_M-
>Timing.clockTick1+
         GravitySeparator_M-
>Timing.clockTickH1* 4294967296.0)) ) *
       (DBL_EPSILON))) {
     rtmSetErrorStatus(GravitySeparator_M,
"Simulation finished");
    }
   }

rt_ertODEUpdateContinuousStates(&GravitySepar
ator_M->solverInfo);
   /* Update absolute time for base rate */
   /* The "clockTick0" counts the number of times
the code of this task has
    * been executed. The absolute time is the
multiplication of "clockTick0"
    * and "Timing.stepSize0". Size of "clockTick0"
ensures timer will not
    * overflow during the application lifespan
selected.
    * Timer of this task consists of two 32 bit
unsigned integers.
```

```
   * The two integers represent the low bits
Timing.clockTick0 and the high bits
   * Timing.clockTickH0. When the low bit
overflows to 0, the high bits increment.
   */
  if                    (!(++GravitySeparator_M-
>Timing.clockTick0)) {
    ++GravitySeparator_M->Timing.clockTickH0;
  }
  GravitySeparator_M->Timing.t[0]            =
rtsiGetSolverStopTime
    (&GravitySeparator_M->solverInfo);
  {
    /* Update absolute timer for sample time: [1.0s,
0.0s] */
    /* The "clockTick1" counts the number of times
the code of this task has
     * been executed. The resolution of this integer
timer is 1.0, which is the step size
     * of the task. Size of "clockTick1" ensures timer
will not overflow during the
     * application lifespan selected.
     * Timer of this task consists of two 32 bit
unsigned integers.
     * The two integers represent the low bits
Timing.clockTick1 and the high bits
     * Timing.clockTickH1. When the low bit
overflows to 0, the high bits increment.
     */
    GravitySeparator_M->Timing.clockTick1++;
    if (!GravitySeparator_M->Timing.clockTick1) {
     GravitySeparator_M-
>Timing.clockTickH1++;
    }
  }
 }                          /* end MajorTimeStep */
}
/* Derivatives for root system: '<Root>' */
void GravitySeparator_derivatives(void)
{
 XDot_GravitySeparator_T *_rtXdot;
 _rtXdot   =   ((XDot_GravitySeparator_T   *)
GravitySeparator_M->derivs);
 /* Derivatives for Integrator: '<S6>/Integrator' */
 _rtXdot->Integrator_CSTATE                  =
GravitySeparator_B.Divide;
 /* Derivatives for Integrator: '<S6>/Integrator1' */
 _rtXdot->Integrator1_CSTATE                 =
GravitySeparator_B.Divide1;
 /* Derivatives for Integrator: '<S6>/Integrator2' */
 _rtXdot->Integrator2_CSTATE                 =
GravitySeparator_B.Divide2;
 /* Derivatives for Integrator: '<S8>/Integrator' */
 _rtXdot->Integrator_CSTATE_a                =
GravitySeparator_B.IntegralGain;
 /* Derivatives for Integrator: '<S6>/Integrator3' */
 _rtXdot->Integrator3_CSTATE                 =
GravitySeparator_B.Divide3;
 /* Derivatives for Integrator: '<S6>/Integrator6' */
 _rtXdot->Integrator6_CSTATE                 =
GravitySeparator_B.Divide4;
 /* Derivatives for Integrator: '<S6>/Integrator7' */
 _rtXdot->Integrator7_CSTATE                 =
GravitySeparator_B.Divide5;
}
/* Model initialize function */
void GravitySeparator_initialize(void)
{
 /* Registration code */
 /* initialize non-finites */
 rt_InitInfAndNaN(sizeof(real_T));
 /* initialize real-time model */
 (void) memset((void *)GravitySeparator_M, 0,

sizeof(RT_MODEL_GravitySeparator_T));
 {
  /* Setup solver object */
  rtsiSetSimTimeStepPtr(&GravitySeparator_M-
>solverInfo,
              &GravitySeparator_M-
>Timing.simTimeStep);
  rtsiSetTPtr(&GravitySeparator_M->solverInfo,
&rtmGetTPtr(GravitySeparator_M));
  rtsiSetStepSizePtr(&GravitySeparator_M-
>solverInfo,
             &GravitySeparator_M-
>Timing.stepSize0);
  rtsiSetdXPtr(&GravitySeparator_M->solverInfo,
&GravitySeparator_M->derivs);
  rtsiSetContStatesPtr(&GravitySeparator_M-
>solverInfo, (real_T **)
              &GravitySeparator_M-
>contStates);
  rtsiSetNumContStatesPtr(&GravitySeparator_M-
>solverInfo,
    &GravitySeparator_M->Sizes.numContStates);

rtsiSetNumPeriodicContStatesPtr(&GravitySeparat
or_M->solverInfo,
    &GravitySeparator_M-
>Sizes.numPeriodicContStates);

rtsiSetPeriodicContStateIndicesPtr(&GravitySepara
tor_M->solverInfo,
    &GravitySeparator_M-
>periodicContStateIndices);

rtsiSetPeriodicContStateRangesPtr(&GravitySepara
tor_M->solverInfo,
    &GravitySeparator_M-
>periodicContStateRanges);
  rtsiSetErrorStatusPtr(&GravitySeparator_M-
>solverInfo, (&rtmGetErrorStatus
    (GravitySeparator_M)));
  rtsiSetRTModelPtr(&GravitySeparator_M-
>solverInfo, GravitySeparator_M);
 }
 rtsiSetSimTimeStep(&GravitySeparator_M-
>solverInfo, MAJOR_TIME_STEP);
```

```
  GravitySeparator_M->intgData.y                    =
GravitySeparator_M->odeY;
  GravitySeparator_M->intgData.f[0]                 =
GravitySeparator_M->odeF[0];
  GravitySeparator_M->intgData.f[1]                 =
GravitySeparator_M->odeF[1];
  GravitySeparator_M->intgData.f[2]                 =
GravitySeparator_M->odeF[2];
  GravitySeparator_M->contStates                    =
((X_GravitySeparator_T *) &GravitySeparator_X);
  rtsiSetSolverData(&GravitySeparator_M-
>solverInfo, (void *)
                    &GravitySeparator_M->intgData);
  rtsiSetSolverName(&GravitySeparator_M-
>solverInfo,"ode3");
  rtmSetTPtr(GravitySeparator_M,
&GravitySeparator_M->Timing.tArray[0]);
  rtmSetTFinal(GravitySeparator_M, 50000.0);
  GravitySeparator_M->Timing.stepSize0 = 1.0;
  rtmSetFirstInitCond(GravitySeparator_M, 1);
  /* Setup for data logging */
  {
    static RTWLogInfo rt_DataLoggingInfo;
    rt_DataLoggingInfo.loggingInterval = NULL;
    GravitySeparator_M->rtwLogInfo              =
&rt_DataLoggingInfo;
  }
  /* Setup for data logging */
  {
    rtliSetLogXSignalInfo(GravitySeparator_M-
>rtwLogInfo, (NULL));
    rtliSetLogXSignalPtrs(GravitySeparator_M-
>rtwLogInfo, (NULL));
    rtliSetLogT(GravitySeparator_M->rtwLogInfo,
"tout");
    rtliSetLogX(GravitySeparator_M->rtwLogInfo,
"");
    rtliSetLogXFinal(GravitySeparator_M-
>rtwLogInfo, "");

rtliSetLogVarNameModifier(GravitySeparator_M-
>rtwLogInfo, "rt_");
    rtliSetLogFormat(GravitySeparator_M-
>rtwLogInfo, 4);
    rtliSetLogMaxRows(GravitySeparator_M-
>rtwLogInfo, 0);
    rtliSetLogDecimation(GravitySeparator_M-
>rtwLogInfo, 1);
    rtliSetLogY(GravitySeparator_M->rtwLogInfo,
"");
    rtliSetLogYSignalInfo(GravitySeparator_M-
>rtwLogInfo, (NULL));
    rtliSetLogYSignalPtrs(GravitySeparator_M-
>rtwLogInfo, (NULL));
  }
  /* block I/O */
  (void) memset(((void *) &GravitySeparator_B), 0,
                sizeof(B_GravitySeparator_T));
  /* states (continuous) */
  {
    (void) memset((void *)&GravitySeparator_X, 0,
                  sizeof(X_GravitySeparator_T));
  }
  /* states (dwork) */
  (void) memset((void *)&GravitySeparator_DW, 0,
                sizeof(DW_GravitySeparator_T));
  /* Matfile logging */

rt_StartDataLoggingWithStartTime(GravitySeparat
or_M->rtwLogInfo, 0.0,
    rtmGetTFinal(GravitySeparator_M),
GravitySeparator_M->Timing.stepSize0,
    (&rtmGetErrorStatus(GravitySeparator_M)));

G_PoissonIntegerGenerator_Start(&GravitySeparat
or_DW.PoissonIntegerGenerator);

G_PoissonIntegerGenerator_Start(&GravitySeparat
or_DW.PoissonIntegerGenerator_p);/*   Start   for
Constant: '<S6>/hwater' */
  GravitySeparator_B.hwater                        =
GravitySeparator_P.hwater_Value;
  /* Start for Constant: '<S6>/hemulsion' */
  GravitySeparator_B.hemulsion                     =
GravitySeparator_P.hemulsion_Value;
  /* Start for Constant: '<S6>/hoil' */
  GravitySeparator_B.hoil                          =
GravitySeparator_P.hoil_Value;
  /* Start for Constant: '<S6>/Ewater' */
  GravitySeparator_B.Ewater                        =
GravitySeparator_P.Ewater_Value;
  /* Start for Constant: '<S6>/Eemulsion' */
  GravitySeparator_B.Eemulsion                     =
GravitySeparator_P.Eemulsion_Value;
  /* Start for Constant: '<S6>/Eoil' */
  GravitySeparator_B.Eoil                          =
GravitySeparator_P.Eoil_Value;
  /* Start for ToWorkspace: '<Root>/To Workspace'
*/
  {
    static int_T rt_ToWksWidths[] = { 1 };
    static int_T rt_ToWksNumDimensions[] = { 1 };
    static int_T rt_ToWksDimensions[] = { 1 };
    static boolean_T rt_ToWksIsVarDims[] = (DNV-
RP-A203);
    static void *rt_ToWksCurrSigDims[] = { (NULL)
};
    static int_T rt_ToWksCurrSigDimsSize[] = { 4 };
    static BuiltInDTypeId rt_ToWksDataTypeIds[] =
{ SS_DOUBLE };
    static int_T rt_ToWksComplexSignals[] = (DNV-
RP-A203);

    static int_T rt_ToWksFrameData[] = (DNV-RP-
A203);
    static const char_T *rt_ToWksLabels[] = { "" };
    static RTWLogSignalInfo rt_ToWksSignalInfo =
{
      1,
      rt_ToWksWidths,
      rt_ToWksNumDimensions,
      rt_ToWksDimensions,
      rt_ToWksIsVarDims,
      rt_ToWksCurrSigDims,
      rt_ToWksCurrSigDimsSize,
      rt_ToWksDataTypeIds,
      rt_ToWksComplexSignals,
      rt_ToWksFrameData,
      { rt_ToWksLabels },
      (NULL),
      (NULL),
      (NULL),
      { (NULL) },
      { (NULL) },
      (NULL),
      (NULL)
    };
    static const char_T rt_ToWksBlockName[] =
"GravitySeparator/To Workspace";

GravitySeparator_DW.ToWorkspace_PWORK.Log
gedData = rt_CreateStructLogVar(
      GravitySeparator_M->rtwLogInfo,
      0.0,
      rtmGetTFinal(GravitySeparator_M),
      GravitySeparator_M->Timing.stepSize0,
      (&rtmGetErrorStatus(GravitySeparator_M)),
      "Foil",
      1,
      0,
      1,
      1.0,
      &rt_ToWksSignalInfo,
      rt_ToWksBlockName);
    if
(GravitySeparator_DW.ToWorkspace_PWORK.Lo
ggedData == (NULL))
      return;
  }
  /*  Start  for  ToWorkspace:  '<Root>/To
Workspace1' */
  {
    static int_T rt_ToWksWidths[] = { 1 };
    static int_T rt_ToWksNumDimensions[] = { 1 };
    static int_T rt_ToWksDimensions[] = { 1 };
    static boolean_T rt_ToWksIsVarDims[] = (DNV-
RP-A203);
    static void *rt_ToWksCurrSigDims[] = { (NULL)
};
    static int_T rt_ToWksCurrSigDimsSize[] = { 4 };
    static BuiltInDTypeId rt_ToWksDataTypeIds[] =
{ SS_DOUBLE };
    static int_T rt_ToWksComplexSignals[] = (DNV-
RP-A203);
    static int_T rt_ToWksFrameData[] = (DNV-RP-
A203);

    static const char_T *rt_ToWksLabels[] = { "" };
    static RTWLogSignalInfo rt_ToWksSignalInfo =
{
      1,
      rt_ToWksWidths,
      rt_ToWksNumDimensions,
      rt_ToWksDimensions,
      rt_ToWksIsVarDims,
      rt_ToWksCurrSigDims,
      rt_ToWksCurrSigDimsSize,
      rt_ToWksDataTypeIds,
      rt_ToWksComplexSignals,
      rt_ToWksFrameData,
      { rt_ToWksLabels },
      (NULL),
      (NULL),
      (NULL),
      { (NULL) },
      { (NULL) },
      (NULL),
      (NULL)
    };
    static const char_T rt_ToWksBlockName[] =
"GravitySeparator/To Workspace1";

GravitySeparator_DW.ToWorkspace1_PWORK.Lo
ggedData = rt_CreateStructLogVar(
      GravitySeparator_M->rtwLogInfo,
      0.0,
      rtmGetTFinal(GravitySeparator_M),
      GravitySeparator_M->Timing.stepSize0,
      (&rtmGetErrorStatus(GravitySeparator_M)),
      "Effectiveness",
      1,
      0,
      1,
      1.0,
      &rt_ToWksSignalInfo,
      rt_ToWksBlockName);
    if
(GravitySeparator_DW.ToWorkspace1_PWORK.L
oggedData == (NULL))
      return;
  }
  /*  Start  for  ToWorkspace:  '<Root>/To
Workspace2' */
  {
    static int_T rt_ToWksWidths[] = { 1 };
    static int_T rt_ToWksNumDimensions[] = { 1 };
    static int_T rt_ToWksDimensions[] = { 1 };
    static boolean_T rt_ToWksIsVarDims[] = (DNV-
RP-A203);
    static void *rt_ToWksCurrSigDims[] = { (NULL)
};
    static int_T rt_ToWksCurrSigDimsSize[] = { 4 };
    static BuiltInDTypeId rt_ToWksDataTypeIds[] =
{ SS_DOUBLE };
    static int_T rt_ToWksComplexSignals[] = (DNV-
RP-A203);
    static int_T rt_ToWksFrameData[] = (DNV-RP-
A203);
    static const char_T *rt_ToWksLabels[] = { "" };
  static RTWLogSignalInfo rt_ToWksSignalInfo =
{
    1,
    rt_ToWksWidths,
    rt_ToWksNumDimensions,
    rt_ToWksDimensions,
    rt_ToWksIsVarDims,
    rt_ToWksCurrSigDims,
    rt_ToWksCurrSigDimsSize,
    rt_ToWksDataTypeIds,
    rt_ToWksComplexSignals,
    rt_ToWksFrameData,
    { rt_ToWksLabels },
    (NULL),
    (NULL),
    (NULL),
    { (NULL) },
    { (NULL) },
    (NULL),
    (NULL)
  };
  static const char_T rt_ToWksBlockName[] =
"GravitySeparator/To Workspace2";

GravitySeparator_DW.ToWorkspace2_PWORK.Lo
ggedData = rt_CreateStructLogVar(
    GravitySeparator_M->rtwLogInfo,
    0.0,
    rtmGetTFinal(GravitySeparator_M),
    GravitySeparator_M->Timing.stepSize0,
    (&rtmGetErrorStatus(GravitySeparator_M)),
    "Fwater",
    1,
    0,
    1,
    1.0,
    &rt_ToWksSignalInfo,
    rt_ToWksBlockName);
  if
(GravitySeparator_DW.ToWorkspace2_PWORK.L
oggedData == (NULL))
    return;
 }
 /*   Start   for   ToWorkspace:   '<Root>/To
Workspace3' */
 {
  int_T dimensions[2] = { 1, 1 };

GravitySeparator_DW.ToWorkspace3_PWORK.Lo
ggedData = rt_CreateLogVar(
    GravitySeparator_M->rtwLogInfo,
    0.0,
    rtmGetTFinal(GravitySeparator_M),
    GravitySeparator_M->Timing.stepSize0,
    (&rtmGetErrorStatus(GravitySeparator_M)),
    "D_actuator",
    SS_DOUBLE,
    0,
    0,
    1,
    1,
    2,
    dimensions,
    NO_LOGVALDIMS,
    (NULL),
    (NULL),
    0,
    1,
    1.0,
    1);
  if
(GravitySeparator_DW.ToWorkspace3_PWORK.L
oggedData == (NULL))
    return;
 }
 /*   Start   for   ToWorkspace:   '<Root>/To
Workspace4' */
 {
  static int_T rt_ToWksWidths[] = { 3 };
  static int_T rt_ToWksNumDimensions[] = { 1 };
  static int_T rt_ToWksDimensions[] = { 3 };
  static boolean_T rt_ToWksIsVarDims[] = (DNV-
RP-A203);
  static void *rt_ToWksCurrSigDims[] = { (NULL)
};
  static int_T rt_ToWksCurrSigDimsSize[] = { 4 };
  static BuiltInDTypeId rt_ToWksDataTypeIds[] =
{ SS_DOUBLE };
  static int_T rt_ToWksComplexSignals[] = (DNV-
RP-A203);
  static int_T rt_ToWksFrameData[] = (DNV-RP-
A203);
  static const char_T *rt_ToWksLabels[] = { "" };
  static RTWLogSignalInfo rt_ToWksSignalInfo =
{
    1,
    rt_ToWksWidths,
    rt_ToWksNumDimensions,
    rt_ToWksDimensions,
    rt_ToWksIsVarDims,
    rt_ToWksCurrSigDims,
    rt_ToWksCurrSigDimsSize,
    rt_ToWksDataTypeIds,
    rt_ToWksComplexSignals,
    rt_ToWksFrameData,
    { rt_ToWksLabels },
    (NULL),
    (NULL),
    (NULL),
    { (NULL) },
    { (NULL) },
    (NULL),
    (NULL)
  };
  static const char_T rt_ToWksBlockName[] =
"GravitySeparator/To Workspace4";

GravitySeparator_DW.ToWorkspace4_PWORK.Lo
ggedData = rt_CreateStructLogVar(
    GravitySeparator_M->rtwLogInfo,
      0.0,
      rtmGetTFinal(GravitySeparator_M),
      GravitySeparator_M->Timing.stepSize0,
      (&rtmGetErrorStatus(GravitySeparator_M)),
      "Oil_cuts",
      1,
      0,
      1,
      1.0,
      &rt_ToWksSignalInfo,
      rt_ToWksBlockName);
    if
(GravitySeparator_DW.ToWorkspace4_PWORK.L
oggedData == (NULL))
      return;
  }
  /*     Start     for     ToWorkspace:     '<Root>/To
Workspace5' */
  {
    static int_T rt_ToWksWidths[] = { 3 };
    static int_T rt_ToWksNumDimensions[] = { 1 };
    static int_T rt_ToWksDimensions[] = { 3 };
    static boolean_T rt_ToWksIsVarDims[] = (DNV-
RP-A203);
    static void *rt_ToWksCurrSigDims[] = { (NULL)
};
    static int_T rt_ToWksCurrSigDimsSize[] = { 4 };
    static BuiltInDTypeId rt_ToWksDataTypeIds[] =
{ SS_DOUBLE };
    static int_T rt_ToWksComplexSignals[] = (DNV-
RP-A203);
    static  int_T  rt_ToWksFrameData[]  =  (DNV-RP-
A203);
    static const char_T *rt_ToWksLabels[] = { "" };
    static RTWLogSignalInfo rt_ToWksSignalInfo =
{
      1,
      rt_ToWksWidths,
      rt_ToWksNumDimensions,
      rt_ToWksDimensions,
      rt_ToWksIsVarDims,
      rt_ToWksCurrSigDims,
      rt_ToWksCurrSigDimsSize,
      rt_ToWksDataTypeIds,
      rt_ToWksComplexSignals,
      rt_ToWksFrameData,
      { rt_ToWksLabels },
      (NULL),
      (NULL),
      (NULL),
      { (NULL) },
      { (NULL) },
      (NULL),
      (NULL)
    };
    static   const   char_T   rt_ToWksBlockName[]   =
"GravitySeparator/To Workspace5";

GravitySeparator_DW.ToWorkspace5_PWORK.Lo
ggedData = rt_CreateStructLogVar(
      GravitySeparator_M->rtwLogInfo,
      0.0,
      rtmGetTFinal(GravitySeparator_M),
      GravitySeparator_M->Timing.stepSize0,
      (&rtmGetErrorStatus(GravitySeparator_M)),
      "Levels",
      1,
      0,
      1,
      1.0,
      &rt_ToWksSignalInfo,
      rt_ToWksBlockName);
    if
(GravitySeparator_DW.ToWorkspace5_PWORK.L
oggedData == (NULL))
      return;
  }
  /*     Start     for     ToWorkspace:     '<Root>/To
Workspace6' */
  {
    int_T dimensions[2] = { 1, 1 };


GravitySeparator_DW.ToWorkspace6_PWORK.Lo
ggedData = rt_CreateLogVar(
      GravitySeparator_M->rtwLogInfo,
      0.0,
      rtmGetTFinal(GravitySeparator_M),
      GravitySeparator_M->Timing.stepSize0,
      (&rtmGetErrorStatus(GravitySeparator_M)),
      "D_valve",
      SS_DOUBLE,
      0,
      0,
      1,
      1,
      2,
      dimensions,
      NO_LOGVALDIMS,
      (NULL),
      (NULL),
      0,
      1,
      1.0,
      1);
    if
(GravitySeparator_DW.ToWorkspace6_PWORK.L
oggedData == (NULL))
      return;
  }
  /* Start for ToWorkspace: '<S6>/To Workspace7'
*/
  {
    int_T dimensions[1] = { 1 };

GravitySeparator_DW.ToWorkspace7_PWORK.Lo
ggedData = rt_CreateLogVar(
      GravitySeparator_M->rtwLogInfo,
      0.0,
      rtmGetTFinal(GravitySeparator_M),
    GravitySeparator_M->Timing.stepSize0,
    (&rtmGetErrorStatus(GravitySeparator_M)),
    "Ewater",
    SS_DOUBLE,
    0,
    0,
    0,
    1,
    1,
    dimensions,
    NO_LOGVALDIMS,
    (NULL),
    (NULL),
    0,
    1,
    1.0,
    1);
  if
(GravitySeparator_DW.ToWorkspace7_PWORK.L
oggedData == (NULL))
    return;
 }
 {
  uint32_T tseed;
  int32_T r;
  int32_T t;
  real_T tmp;
  /*          InitializeConditions          for
UniformRandomNumber:  '<S3>/Random  Number'
*/
  tmp                                      =
floor(GravitySeparator_P.RandomNumber_Seed);
  if (rtIsNaN(tmp) || rtIsInf(tmp)) {
    tmp = 0.0;
  } else {
    tmp = fmod(tmp, 4.294967296E+9);
  }
  tseed    =    tmp    <    0.0    ?    (uint32_T)-
(int32_T)(uint32_T)-tmp : (uint32_T)tmp;
  r = (int32_T)(tseed >> 16U);
  t = (int32_T)(tseed & 32768U);
  tseed = ((((tseed - ((uint32_T)r << 16U)) + t) <<
16U) + t) + r;
  if (tseed < 1U) {
    tseed = 1144108930U;
  } else {
    if (tseed > 2147483646U) {
      tseed = 2147483646U;
    }
  }
  GravitySeparator_DW.RandSeed = tseed;

GravitySeparator_DW.RandomNumber_NextOutp
ut =

(GravitySeparator_P.RandomNumber_Maximum -

GravitySeparator_P.RandomNumber_Minimum)  *
rt_urand_Upu32_Yd_f_pw_snf
      (&GravitySeparator_DW.RandSeed)          +
GravitySeparator_P.RandomNumber_Minimum;
  /*    End    of    InitializeConditions    for
UniformRandomNumber:  '<S3>/Random  Number'
*/
  /*     InitializeConditions     for     S-Function
(sdspcumsumprod): '<S3>/Cumulative Sum' */

GravitySeparator_DW.CumulativeSum_RunningC
umVal = 0.0;
  /*          InitializeConditions          for
UniformRandomNumber:  '<S4>/Random  Number'
*/
  tmp                                      =
floor(GravitySeparator_P.RandomNumber_Seed_d
);
  if (rtIsNaN(tmp) || rtIsInf(tmp)) {
    tmp = 0.0;
  } else {
    tmp = fmod(tmp, 4.294967296E+9);
  }
  tseed    =    tmp    <    0.0    ?    (uint32_T)-
(int32_T)(uint32_T)-tmp : (uint32_T)tmp;
  r = (int32_T)(tseed >> 16U);
  t = (int32_T)(tseed & 32768U);
  tseed = ((((tseed - ((uint32_T)r << 16U)) + t) <<
16U) + t) + r;
  if (tseed < 1U) {
    tseed = 1144108930U;
  } else {
    if (tseed > 2147483646U) {
      tseed = 2147483646U;
    }
  }
  GravitySeparator_DW.RandSeed_b = tseed;

GravitySeparator_DW.RandomNumber_NextOutp
ut_d =

(GravitySeparator_P.RandomNumber_Maximum_
m -

GravitySeparator_P.RandomNumber_Minimum_c)
* rt_urand_Upu32_Yd_f_pw_snf
      (&GravitySeparator_DW.RandSeed_b) +

GravitySeparator_P.RandomNumber_Minimum_c;
  /*    End    of    InitializeConditions    for
UniformRandomNumber:  '<S4>/Random  Number'
*/
  /*     InitializeConditions     for     S-Function
(sdspcumsumprod): '<S4>/Cumulative Sum' */

GravitySeparator_DW.CumulativeSum_RunningC
umVal_b = 0.0;
  /*     InitializeConditions     for     Integrator:
'<S6>/Integrator' incorporates:
   *      InitializeConditions     for     Integrator:
'<S6>/Integrator1'
   */
  if (rtmIsFirstInitCond(GravitySeparator_M)) {
    GravitySeparator_X.Integrator_CSTATE = 1.56;
    GravitySeparator_X.Integrator1_CSTATE = 1.67;
  }
  GravitySeparator_DW.Integrator_IWORK = 1;
  /* End of InitializeConditions for Integrator: '<S6>/Integrator' */
  /* InitializeConditions for Integrator: '<S6>/Integrator1' */
  GravitySeparator_DW.Integrator1_IWORK = 1;
  /* InitializeConditions for Integrator: '<S6>/Integrator2' incorporates:
   * InitializeConditions for Integrator: '<S6>/Integrator3'
   */
  if (rtmIsFirstInitCond(GravitySeparator_M)) {
    GravitySeparator_X.Integrator2_CSTATE = 3.21;
    GravitySeparator_X.Integrator3_CSTATE = 0.1;
  }
  GravitySeparator_DW.Integrator2_IWORK = 1;
  /* End of InitializeConditions for Integrator: '<S6>/Integrator2' */
  /* InitializeConditions for Integrator: '<S8>/Integrator' */
  GravitySeparator_X.Integrator_CSTATE_a = GravitySeparator_P.Integrator_IC;
  /* InitializeConditions for Integrator: '<S6>/Integrator3' */
  GravitySeparator_DW.Integrator3_IWORK = 1;
  /* InitializeConditions for Integrator: '<S6>/Integrator6' incorporates:
   * InitializeConditions for Integrator: '<S6>/Integrator7'
   */
  if (rtmIsFirstInitCond(GravitySeparator_M)) {
    GravitySeparator_X.Integrator6_CSTATE = 0.3;
    GravitySeparator_X.Integrator7_CSTATE = 0.9;
  }
  GravitySeparator_DW.Integrator6_IWORK = 1;

  /* End of InitializeConditions for Integrator: '<S6>/Integrator6' */
  /* InitializeConditions for Integrator: '<S6>/Integrator7' */
  GravitySeparator_DW.Integrator7_IWORK = 1;
  /* SystemInitialize for Chart: '<Root>/Effectiveness' */

GravitySeparator_DW.is_active_c8_GravitySeparator = 0U;
  GravitySeparator_DW.is_c8_GravitySeparator = GravitySepar_IN_NO_ACTIVE_CHILD;
  /* set "at time zero" to false */
  if (rtmIsFirstInitCond(GravitySeparator_M)) {
    rtmSetFirstInitCond(GravitySeparator_M, 0);
  }
 }
}
/* Model terminate function */
void GravitySeparator_terminate(void)
{

Gr_PoissonIntegerGenerator_Term(&GravitySeparator_DW.PoissonIntegerGenerator);

Gr_PoissonIntegerGenerator_Term(&GravitySeparator_DW.PoissonIntegerGenerator_p);
}
```

SIMULINK C/ DATA

```
/*
 * GravitySeparator_data.c
* Academic License - for use in teaching, academic research, and meeting
 * course requirements at degree granting institutions only.  Not for
 * government, commercial, or other organizational use.
* Code generation for model "GravitySeparator".
* Model version              : 1.84
 * Simulink Coder version : 8.11 (R2016b) 25-Aug-2016
 * C source code generated on : Sat Jun 10 18:23:55 2017
* Target selection: grt.tlc
 * Note: GRT includes extra infrastructure and instrumentation for prototyping
 * Embedded hardware selection: Intel->x86-64 (Windows64)
 * Code generation objectives: Unspecified
 * Validation result: Not run
 */
#include "GravitySeparator.h"
#include "GravitySeparator_private.h"
/* Block parameters (auto storage) */
P_GravitySeparator_T GravitySeparator_P = {
  98.8652555812759,           /* Mask Parameter: PIDController_I
                               * Referenced by: '<S8>/Integral Gain'
                               */
  5648.86645030705,           /* Mask Parameter: PIDController_P
                               * Referenced by: '<S8>/Proportional Gain'
                               */
  0.0,                        /* Expression: 0
```

```
                                       * Referenced by: '<S3>/Random Number'
                                       */
  0.003,                              /* Expression: 0.003
                                       * Referenced by: '<S3>/Random Number'
                                       */
  1125.0,                             /* Expression: 1125
                                       * Referenced by: '<S3>/Random Number'
                                       */
  0.0,                                /* Expression: 0
                                       * Referenced by: '<S4>/Random Number'
                                       */
  0.005,                              /* Expression: 0.005
                                       * Referenced by: '<S4>/Random Number'
                                       */
  1125.0,                             /* Expression: 1125
                                       * Referenced by: '<S4>/Random Number'
                                       */
  0.0081,                             /* Expression: 0.0081
                                       * Referenced by: '<S7>/Kv des (theoretical)'
                                       */
  1.0,                                /* Expression: 1
                                       * Referenced by: '<Root>/Constant'
                                       */
  1.56,                               /* Expression: 1.56
                                       * Referenced by: '<S6>/hwater'
                                       */
  1.67,                               /* Expression: 1.67
                                       * Referenced by: '<S6>/hemulsion'
                                       */
  3.21,                               /* Expression: 3.21
                                       * Referenced by: '<S6>/hoil'
                                       */
  61.111111111111114,                 /* Expression: InitialConditionForIntegrator
                                       * Referenced by: '<S8>/Integrator'
                                       */
  0.81,                               /* Expression: 0.81
                                       * Referenced by: '<S7>/Max flow m3//s'
                                       */
  0.1,                                /* Expression: 0.1
                                       * Referenced by: '<S6>/Ewater'
                                       */
  0.3,                                /* Expression: 0.3
                                       * Referenced by: '<S6>/Eemulsion'
                                       */
  0.9,                                /* Expression: 0.9
                                       * Referenced by: '<S6>/Eoil'
                                       */
  0.57735026918962584,                /* Expression: 1/sqrt(3)
                                       * Referenced by: '<S6>/Cd'
                                       */
  3.4641,                             /* Expression: 3.4641
                                       * Referenced by: '<S6>/lweir'
                                       */
  9.81,                               /* Expression: 9.81
                                       * Referenced by: '<S6>/gravity'
                                       */
  3.0,                                /* Expression: 3
                                       * Referenced by: '<S6>/hweir'
                                       */
  2.0,                                /* Expression: 2
                                       * Referenced by: '<S6>/Radius'
                                       */
  1.1,                                /* Expression: 1.1
                                       * Referenced by: '<S6>/Fin'
                                       */
  1.0,                                /* Expression: 1
                                       * Referenced by: '<S6>/Pfup'
                                       */
  1000.0,                             /* Expression: 1000
                                       * Referenced by: '<S6>/Rho_water'
                                       */
  845.0,                              /* Expression: 845
                                       * Referenced by: '<S6>/Rho_oil'
                                       */
  0.0013,                             /* Expression: 1.3E-3
                                       * Referenced by: '<S6>/vis_oil'
                                       */
  0.000547,                           /* Expression: 0.547E-3
                                       * Referenced by: '<S6>/vis_water'
                                       */
  0.0017,                             /* Expression: 1E-3*1.7
                                       * Referenced by: '<S6>/d'
                                       */
  0.55,                               /* Expression: 0.55
                                       * Referenced by: '<S6>/Ein'
                                       */
  10.0,                               /* Expression: 10
                                       * Referenced by: '<S6>/Length'
                                       */
  0.05,                         /* Expression: 0.5*0.1
                               *        Referenced        by:
'<S6>/Pqup'
                               */
  0.0,                         /* Expression: 0
                               *        Referenced        by:
'<S6>/Constant1'
                               */
  2.0,                         /* Expression: 2
                               *        Referenced        by:
'<S14>/Constant1'
                               */
  2.0,                         /* Expression: 2
                               *        Referenced        by:
'<S14>/Constant'
                               */
  1.0,                         /* Expression: 1
                               *        Referenced        by:
'<S6>/Pfdown'
                               */
  0.66666666666666663,                 /* Expression:
2/3
                               *        Referenced        by:
'<S6>/Constant2'
                               */
  0.12345,                        /* Expression: 0.12345
                               *        Referenced        by:
'<S6>/tcoalescence'
                               */
  0.5,                         /* Expression: 0.5
                               *        Referenced        by:
'<S6>/Pqdown'
                               */
  0.0,                         /* Expression: 0
                               *        Referenced        by:
'<S6>/Constant'
                               */
  2.0,                         /* Expression: 2
                               *        Referenced        by:
'<S13>/Constant1'
                               */
  2.0,                         /* Expression: 2
                               *        Referenced        by:
'<S13>/Constant'
                               */
  2.0,                         /* Expression: 2
                               *        Referenced        by:
'<S15>/Constant1'
                               */
  2.0                          /* Expression: 2
                               *        Referenced        by:
'<S15>/Constant'
                               */
};
```